
MICRO-ORDINATEUR
ITT 2020*

MANUEL D'UTILISATION

* Apple System



- Afin de faciliter la compréhension de ce manuel, les textes ont été composés dans deux caractères différents:
l'un pour les textes généraux, l'autre pour faire ressortir les textes qui doivent apparaître sur l'écran.

Exemple:

```
714  YSPEED = -1 * YSPEED
```

```
719  GOTO 706
```

qui apparaîtra sur l'écran de la façon suivante:

```
714  YSPEED=-1*YSPEED
```

```
719  GOTO 706
```

Notez bien la façon de différencier la lettre O du chiffre 0.

- Tout au long de ce manuel, vous verrez des indications de ce genre:

RETURN ou REPT ou encore "tapez 130 puis RETURN,"

B

Cela signifie que vous devez appuyer sur la ou les touches dont les noms sont indiqués dans le ou les cartouches.

Dans l'exemple REPT il faut appuyer sur les deux touches simulta-

nément.

B

SOMMAIRE

CHAPITRE 1	branchements - mise en route	page 5
CHAPITRE 2	le clavier,touches spéciales,utilisation	page 6
CHAPITRE 3	introduction au basic	page 11
CHAPITRE 4	la couleur c'est la vie	page 23
CHAPITRE 5	écrivons notre premier programme	page 27
CHAPITRE 6	modes de comparaison	page 35
CHAPITRE 7	un peu... de logique	page 40
CHAPITRE 8	l'organigramme	page 45
CHAPITRE 9	les boucles (FOR...NEXT...STEP)	page 49
CHAPITRE 10	comment se placer sur l'écran	page 60
CHAPITRE 11	introduction des données (INPUT)	page 64
CHAPITRE 12	les variables alphanumériques,leur dimension	page 68
CHAPITRE 13	les registres,comment les dimensionner	page 81
CHAPITRE 14	les sous-programmes (GOSUB)	page 96
CHAPITRE 15	levier de commande	page 104
CHAPITRE 16	CALL,POKE,PEEK	page 108
CHAPITRE 17	graphiques en couleur à haute résolution	page 116
CHAPITRE 18	glossaire alphabetique des instructions et commandes basic et leur signification	page 128
ANNEXE	programmes divers	page 142

Félicitations!

Vous êtes maintenant l'heureux possesseur d'un "MICRO ORDINATEUR ITT 2020".

Ce manuel a été conçu de la manière la plus simple possible et suivant un ordre logique afin que vous découvriez progressivement les possibilités nombreuses de votre ordinateur personnel, et la manière de vous en servir le mieux possible.

Naturellement, si vous êtes déjà un "initié" en informatique, le contenu de ce manuel vous paraîtra extrêmement simple, mais il vous faudra découvrir de nouvelles manières moins "formelles" d'utiliser la programmation à titre personnel. Si par contre vous n'êtes pas un spécialiste, ce manuel a justement été conçu pour qu'en peu de temps vous deveniez, vous aussi, un "programmeur" confirmé, vous donnant accès au domaine extraordinaire de l'informatique "personnelle".

N'ayez aucune crainte, la compréhension et le maniement sont aisés sous la condition expresse que vous appreniez en utilisant simultanément l'appareil et le manuel.

En effet, si vous ne faites que parcourir ce manuel, sans essayer immédiatement chaque instruction sur votre ordinateur, le résultat sera décevant.

CHAPITRE 1

BRANCHEMENTS-MISE EN ROUTE

Toutes les connexions étant établies, il vous faut effectuer ces opérations:

- 1° - mise en marche de votre TV, en ayant soin de réduire au minimum son volume sonore (en effet, le son ne vous servira pas);
- 2° - mise en marche du micro-ordinateur ITT (par l'interrupteur visible sur le schéma); vous verrez s'allumer, en bas du clavier, le témoin lumineux de mise sous tension (attention ce témoin n'est pas une touche et ne peut être enfoncé);
- 3° - sélectionner un canal UHF (vous référer à la notice de votre récepteur TV, ou demander à votre revendeur TV de vous indiquer le processus, en principe chaîne 2 canal 36).

Notes

CHAPITRE 2

LE CLAVIER, TOUCHES SPECIALES, UTILISATION

Vous êtes maintenant prêt à utiliser votre ordinateur.

EXAMINONS CE QUI SE TROUVE SUR LE CLAVIER :

Tout d'abord, ne tenez pas compte de ce qui a pu apparaître sur votre écran TV!

En premier lieu, appuyez sur la touche **RESET** (située dans le coin supérieur droit du clavier).

Essayez.

Si toutes les opérations précédentes ont été effectuées correctement (voir chapitre 1), votre ordinateur va émettre un son bref (BIP) quand vous relâcherez la touche **RESET**, et l'écran devra présenter un astérisque * dans le coin inférieur à gauche, suivi d'un carré clignotant (que nous appellerons dorénavant CURSEUR). Pour le moment, ne vous occupez pas des signes ou caractères qui emplissent l'écran.

Revenons à notre clavier. Vous y trouverez 2 touches marquées **SHIFT**. Ces touches remplissent la même fonction que les touches MAJUSCULES sur une machine à écrire classique, c'est-à-dire qu'elles permettent l'utilisation d'une seule touche pour 2 caractères. Par exemple, si vous appuyez seulement la touche **N**, vous obtiendrez sur l'écran la lettre N; si vous tenez enfoncée la touche **SHIFT** et appuyez sur la touche **N**, vous obtiendrez le symbole supérieur, soit \wedge .

Familiarisez-vous avec cette touche en l'utilisant.

Il y a seulement deux exceptions à la règle ci-dessus. Elles concernent les touches **M** et **G**. En effet, appuyez sur **SHIFT** et **M** : le symbole qui apparaît (non représenté sur la touche) est une sorte de crochet (]). Appuyez sur **SHIFT** et **G** : vous verrez apparaître non pas le mot BELL mais un simple G. Vous trouverez plus loin l'explication du mot BELL.

- Une autre remarque importante concerne le zéro et la lettre O. Par convention, et pour éviter toute erreur de programmation, le chiffre zéro sera représenté sous le symbole \emptyset au lieu de O. D'ailleurs, regardez votre clavier: il s'agit bien de deux touches distinctes.

A propos, que s'est-il passé sur l'écran, pendant que vous vous exerciez à taper des lettres ou des chiffres?

Il s'est rempli de signes ou de symboles de toutes sortes; ce sont ceux que vous avez vus au début de ce chapitre. Puis tout ce que vous avez tapé s'est inscrit dans le bas de l'écran. Si maintenant vous désirez obtenir un écran vierge, procédez de la manière suivante:

1° - appuyez sur la touche **ESC** (1ère de la 2ème rangée en partant du haut);

2° - appuyez et tenez enfoncée la touche **SHIFT**;

3° - appuyez sur la touche **P** ;

4° - relâchez les touches **SHIFT** et **P** : tout l'écran devient vierge d'inscription, sauf le curseur en haut à gauche;

5° - appuyez sur **RETURN** et l'astérisque réapparaît.

N.B.: tant que le contraire n'est pas spécifié dans le manuel, on doit toujours relâcher la dernière touche tapée avant de frapper la suivante.

- Pour une bonne compréhension des explications de ce manuel relatives à l'utilisation du clavier, vous devez lire attentivement ce qui suit.

Dans les exemples qui suivront, nous visualiserons les mots ou phrases dans leur ordre séquentiel (c'est-à-dire dans l'ordre respectant celui des mots et phrases que vous allez introduire).

UTILISATION SIMULTANÉE DE DEUX TOUCHES :

Prenons un exemple: pour obtenir le signe $\$$, vous devez d'abord presser et tenir enfoncée la touche **SHIFT**, et simultanément appuyer sur

la touche .

Chaque fois qu'une action simultanée sera nécessaire, nous indiquerons les symboles l'un au dessus de l'autre, comme ceci:

SHIFT




Autre exemple (pour rendre l'écran vierge):

SHIFT

ESC **P** **RETURN**

TOUCHES À FONCTIONS PARTICULIÈRES À VOTRE ORDINATEUR :

La touche **CTRL** accomplit des fonctions spéciales, un peu de la même manière que la touche **SHIFT**, mais avec la différence essentielle que les touches tapées simultanément avec **CTRL** ne sont jamais affichées sur l'écran.

Par exemple: appuyez, tenez enfoncé **CTRL** et appuyez sur la touche .

Quand vous appuierez sur les deux touches, votre ordinateur répondra par un son bref (BIP).

Cela signifie, en cours d'utilisation, qu'il désire attirer votre attention sur certaines choses (nous verrons plus loin lesquelles).

Maintenant tapez:



- Vous vous rappelez nos conventions? Cela signifie: appuyez et relâchez la touche **RESET**, puis appuyez et tenez enfoncé **CTRL** et **B**, relâchez, puis appuyez et relâchez **RETURN**.

Le résultat de cette opération fait apparaître sur l'écran (en bas à gauche) le curseur clignotant précédé d'une "pointe de flèche".

(N.B.: si la première fois vous n'obtenez pas ce résultat, recommencez!)
Nous vous dirons plus loin la signification de ceci (voir glossaire page 139).

La touche **REPT**

Comme vous l'avez déjà deviné, cette touche est l'abréviation de répétition. Le fait de la tenir enfoncée simultanément avec n'importe quel caractère du clavier permet d'obtenir la répétition de ce dernier autant de fois que vous le désirez, aussi longtemps que vous tiendrez les deux touches enfoncées.

UNE REMARQUE EN PASSANT :

Si vous avez appuyé sur la touche **RETURN** au cours des manipulations précédentes, l'ordinateur émettra un son bref (BIP), et inscrira sur l'écran le message suivant: "***SYNTAX ERR".

A ce stade de votre initiation, ne vous en préoccupez pas. L'explication viendra plus loin.

Enfin les dernières touches spéciales sont celles qui déplacent le curseur vers la droite  ou vers la gauche  (voir page 12).

Maintenant, avant de continuer, faites des essais, encore des essais, toujours des essais, et tranquillisez vous: rien de ce que vous pourrez taper sur votre clavier n'est susceptible d'endommager l'appareil!



Vous devez avoir maintenant sur l'écran une "pointe de flèche" avec la curseur clignotant à sa droite, et rien d'autre ensuite. (Au cas où vous auriez tapé quelque chose d'indéterminé appuyez sur la touche **RETURN**).

Vous êtes donc dans les conditions requises pour que votre ordinateur exécute vos ordres. Nous vous donnerons au prochain chapitre des détails sur le langage que comprend l'ordinateur (appelé: le BASIC), mais pour le moment faisons quelques essais de commandes simples, exécutées en mode d'affichage immédiat.

Vous allez taper les mots ci-dessous, exactement comme ils sont écrits:

PRINT "BONJOUR" (c'est-à-dire **P R I N T** **SHIFT** **2** **B O N J O U R** **SHIFT** **2**)

Appuyez sur **RETURN** . Que se passe-t-il?

BONJOUR est affiché sur l'écran et si jamais vous avez commis une erreur quelconque (par exemple tapé **PRINTE** ou omis un ") l'écran affichera *****SYNTAX ERR**". Si tel est le cas, recommencez en faisant plus attention!

Vous pouvez essayer d'inscrire un ou plusieurs mots ou bien une phrase de votre cru!

Attention, tapez **PRINT** puis placez toujours ce que vous voulez afficher entre guillemets "..."; et n'oubliez pas **RETURN** pour voir votre texte s'inscrire sur l'écran. La dernière condition à respecter est de ne pas inclure entre les guillemets plus de 122 caractères ou espaces, sinon l'écran affichera:

*****TOO LONG ERR**

Vous pouvez aussi, emporté par votre enthousiasme, essayer d'entrer en une seule fois un chapitre entier de M. Paul CLAUDEL. Dans une certaine mesure votre ordinateur aura pitié de vous: c'est-à-dire qu'à partir de 240 caractères, donc avant que vous ayez placé le 2ème guillemet, il émettra un BIP et passera à la ligne en affichant de nouveau le >□. Vous n'aurez plus alors qu'à recommencer en tenant compte des instructions indiquées plus haut.

Essayez...

Bien sûr, vous avez parfaitement réussi la première fois, mais nous devons vous indiquer une autre source d'erreur possible: si vous oubliez les 2 guillemets, l'ordinateur ne prendra pas en compte ce que vous avez écrit après **PRINT** et répondra par ∅.

APRÈS LES LETTRES... LES CHIFFRES :

Essayez ceci:

PRINT "150" **RETURN** , et sur l'écran vous lirez...150

Vous avez déjà compris que toute inscription, quelle qu'elle soit, précédée de la commande **PRINT** et comprise entre les guillemets, sera affichée sur l'écran.

Tapez: PRINT 150

Bien que le chiffre ne soit pas entre guillemets, l'écran affichera 150. En fait, tout nombre compris entre les valeurs maximales autorisées par votre ordinateur (-32767 à +32767) peut être affiché sans problème et sans guillemets. A ce sujet, votre ordinateur peut évidemment vous servir de calculatrice. Nous verrons au chapitre suivant comment tirer le maximum des possibilités "calculatrice" de votre ordinateur.

Exemple: PRINT 15283 + 7675 RETURN

vous lisez la réponse: 22958

Nous pensons que maintenant il serait temps de vous initier au langage (BASIC) qui vous permettra d'interroger l'ordinateur. Progressivement, par des exemples choisis, nous vous apprendrons à faire exécuter par la machine des programmes de plus en plus sophistiqués.

- Remarque: n'écrivez jamais du texte comme ceci:

```
PRINT "LE LIEU-DIT DE "BASTIDE BLANCHE" A DISPARU"
```

En effet, l'ordinateur reconnaissant quatre guillemets (c'est-à-dire deux de trop) vous indiquera:

```
***SYNTAX ERR
```

Dans les exemples précédents, vous avez vu que votre ordinateur ITT fait la différence entre le texte (par exemple PRINT "BONJOUR, ME VOILA") et le numérique (par exemple PRINT 123 + 154). Il se contente en effet d'afficher le premier alors qu'il calcule le second.

- La définition suivante est valable pour tous les autres chapitres:

Aucun calcul numérique ne peut se faire pour une expression entre guillemets, car l'ordinateur la considérerait alors uniquement comme du texte.

CHAPITRE 3

INTRODUCTION AU BASIC

SIGNIFICATION DE LA TOUCHE "RETURN":

A partir de maintenant, nous ne vous indiquerons plus quand il faut taper `RETURN`. Souvenez-vous simplement de sa signification:

- 1° la touche `RETURN` sert à introduire dans la mémoire de l'ordinateur ce que vous avez inscrit sur l'écran (donc à prendre en compte le texte écrit entre le `RETURN` précédent et celui que vous allez frapper).

DESCRIPTION

Il existe en informatique des langages conventionnels dont un des plus simples est le "BASIC". Celui-ci est composé d'un certain nombre de mots et de caractères acceptés par la machine, sous condition que leur orthographe et ponctuation soient respectées.

Vous trouverez, en fin de manuel, un glossaire exhaustif de ce langage.

En fait, vous avez déjà utilisé, au chapitre précédent, une commande BASIC! Eh oui...`PRINT` en est une, ainsi que... les guillemets.

Nous allons maintenant examiner plus avant les différentes possibilités de la commande `PRINT`, utilisée avec des ponctuations diverses.

ADDITION:

Comme nous l'avons déjà vu, `PRINT 3 + 4` vous affichera 7. Mais vous pouvez avoir besoin d'afficher les opérateurs de cette addition devant le résultat. Dans ce cas vous procéderez ainsi:

```
PRINT "3 + 4 =" ; 3 + 4
```

Si nous décomposons, nous constatons que la partie entre guillemets (vous vous rappelez?) affichera à la ligne suivante: `3 + 4 =`

Nous trouvons ensuite le point-virgule. Cette ponctuation a une signification extrêmement précise qui indique:

- 1° - à la machine qu'il y a une instruction subdivisée en deux (cf. l'exemple ci-dessus) sur la même ligne
- 2° - indique que le résultat de la 2ème instruction (`3 + 4` soit 7) devra être affiché immédiatement à la suite de la 1ère.
Vous verrez donc affiché:

```
3 + 4 = 7
```

Pour plus de compréhension, essayez ceci:

```
PRINT "3 + 4 =" (puis) RETURN
PRINT 3 + 4      (puis) RETURN
```

Vous obtiendrez:

```
3 + 4 =
7
```


- Avant d'étudier les autres modes de calcul, revenons sur le seul signe dont nous ne vous avons pas donné d'explication dans l'exemple ci-dessus: le signe **+**.

Nous voulons signifier que ce signe est compris par la machine comme un "Opérateur", c'est-à-dire qu'il s'agit là aussi d'une commande BASIC, au même titre que les signes utilisés pour multiplier, diviser ou soustraire. Ne haussez pas les épaules, car cela n'est pas aussi évident! Ne haussez pas les épaules, car cela n'est pas aussi évident!

En effet, en arithmétique scolaire, la multiplication est symbolisée par "X" et la division par ":". En BASIC, l'opérateur utilisé pour la multiplication est un astérisque *****, et pour la division c'est une barre oblique **/**.

UNE REMARQUE POUR VOUS FACILITER L'ÉCRITURE :

Il arrive bien souvent, et surtout pour les programmeurs non familiarisés avec une machine à écrire, de commettre des fautes de frappe. Que peut-on faire pour y remédier, surtout quand on s'en aperçoit à la fin de la phrase?

Par exemple, tapez: PRINT "MICRO ORDINATEUR ITT (sans frapper le second guillemet) et, comme d'habitude, appuyez sur RETURN.

L'écran affichera ***SYNTAX ERR, car nous avons oublié le 2ème guillemet.

Si l'inscription avait été correcte (c'est-à-dire si vous aviez mis le second guillemet à droite), la réponse aurait été MICRO ORDINATEUR ITT.

Maintenant tapez l'instruction suivante où le texte entre guillemets contient une faute d'orthographe:

PRINT "MICRO ORDINATEURE ITT" (non, n'appuyez pas sur **RETURN** !)
Vous remarquerez que le curseur clignotant se trouve immédiatement à la droite du dernier guillemet.

- Pour modifier ORDINATEURE en ORDINATEUR, vous allez utiliser la touche **←**, qui a pour effet, à chaque fois qu'elle est enfoncée, de faire revenir d'une place en arrière le curseur. Appuyez sur cette touche (dans l'exemple: 6 fois) de manière à ce que le curseur clignotant soit sur la lettre E, puis **RETURN**.

Eh oui! encore un ***SYNTAX ERR. La raison est simple.

En effet, quand le curseur revient en arrière, il "efface" les caractères sur lesquels il est passé (dans le cas présent, nous vous avons ramené au 1er exemple de ce chapitre) et ce, non pas de l'écran, mais de la mémoire ordinateur. Afin d'éviter cela, il y a deux solutions:

- 1° - après avoir effectué votre rectification, appuyez sur la touche **→** jusqu'à ce que le curseur dépasse le dernier guillemet (puis **RETURN**);
- 2° - retapez ITT et le guillemet sur le clavier puis **RETURN**.

Essayez tout cela plusieurs fois! (Remarque: en règle générale, pour modifier un caractère il suffit de placer le curseur sur le caractère et de taper la nouvelle lettre).

N.B.: si la phrase inscrite est longue, il est intéressant de conjuguer l'emploi simultané des touches **→** et **←** avec la touche **REPT**.

C'est-à-dire: appuyez une fois sur ←, puis appuyez sur REPT jusqu'à ce que le curseur soit là où vous voulez.

Essayez... rapide n'est-ce pas?

- Vous devez également savoir que dans ce langage BASIC, votre ordinateur n'accepte que des nombres entiers (-27 ou 145). Il refuse les décimales (3,14 ou 18,4).

Il existe, bien sûr, un langage plus évolué qui vous permettra de traiter les nombres décimaux (avec la virgule flottante) et bien d'autres choses!... Il est enregistré séparément sur cassette spéciale et fera l'objet d'un second manuel.

De ce fait, si vous tentez de diviser 27 par 4, l'écran affichera 6 et non pas 6,75, c'est-à-dire le nombre de fois que le diviseur est contenu dans le dividende.

Pour obtenir le reste, il faut se servir de la commande suivante:

MOD (abréviation du terme mathématique "Modulo") et agir de la manière suivante:

PRINT 27 MOD 4, Le résultat affiché sera 3, soit le reste. (Attention: n'utilisez pas MOD sans diviseur ni dividende, l'ordinateur indiquerait une erreur).

Bien entendu, vous pouvez entrer plusieurs opérations sur une même ligne. Par exemple:

PRINT 6 + 4/5 - 1 * 2

Nous verrons, avant la fin de ce chapitre, les règles précises qui gouvernent ces opérations multiples.

ÉLÉVATION À UNE PUISSANCE :

Si on désire multiplier un chiffre par lui-même un certain nombre de fois, on peut taper évidemment:

PRINT 2 * 2 * 2 * 2 * 2

Mais il est infiniment plus simple d'agir comme en mathématiques et utiliser la forme suivante:

PRINT 2 ^ 5

Le symbole ^ est celui de l'élévation à la puissance.

Ce symbole est obtenu en tapant SHIFT

N

Enfin, nous vous rappelons que les nombres autorisés sont ceux compris entre (-) 32767 et (+) 32767.

Dépasser ces limites causera l'affichage du message: ***>32767 ERR

Exemples: PRINT 10 ^ 10 (impossible, le résultat étant supérieur à 32767)

PRINT 84272 (également supérieur à 32767, donc faux)

PRIORITÉ DES OPÉRATEURS ARITHMÉTIQUES :

Dans le cas d'opérations simples sur la même ligne, on ne peut préjuger du résultat si l'on n'a pas établi au préalable une règle des opérations. Par exemple:

$20 + 12/4$ pourrait aussi bien s'exprimer par: $(20 + 12)/4 = 32$
que par: $20 + (12/4) = 23$

(dans l'exemple ci-dessus, le résultat pour l'ordinateur sera 23).

Il y a donc certaines règles que nous allons vous exposer et auxquelles obéit votre ordinateur.

Lorsque vous posez une opération (comme dans l'exemple ci-dessus) sur une feuille de papier, vous obéissez à des règles bien précises pour éviter toute confusion. C'est-à-dire que vous avez la notion de priorité des opérations. Votre ordinateur possède lui aussi cette notion de priorité des opérations. Il obéit aux règles 1°, 2°, 3° et 4° qui suivent. Ces règles sont données dans l'ordre de priorité décroissante (c'est-à-dire que 1° est prioritaire sur 2°, 2° sur 3°, etc...).

- 1° - quand le signe - sert à indiquer un nombre négatif et qu'il est de ce fait placé à sa gauche (ex: $-3 + 2$), l'ordinateur appliquera d'abord le signe (-), ainsi le résultat donnera -1.

Autre exemple: $A = 6$

`PRINT - A + 10` (réponse = 4)

- 2° - après le changement de signe, intervient l'opérateur d'élévation à une puissance. Les opérations se font toujours de gauche à droite ainsi:

$$4 \wedge 2 \wedge 3 = 4096$$

- 3° - ensuite, viennent les opérateurs de multiplication, division et modulo qui ont tous trois une priorité égale mais toujours en opérant de la gauche vers la droite.
- 4° - enfin seront prises en compte les additions et les soustractions (même valeur) et toujours de gauche à droite

Nous pouvons nous résumer ainsi:

- 1° - (les signes "moins" affectant une valeur négative aux chiffres)
2° \wedge (puissance, de gauche à droite)
3° MOD, *, / (modulo, multiplication, division, de gauche à droite)
4° +, -, (addition, soustraction, de gauche à droite).

Nous vous recommandons d'effectuer les opérations suivantes, sans vous servir de l'ordinateur et de comparer ensuite vos résultats avec celui-ci, en recherchant bien entendu pourquoi vous avez obtenu d'éventuelles différences (lorsque vous calculerez les exemples avec votre ordinateur, n'oubliez pas l'ordre PRINT avant les valeurs à calculer).

$$5 + 7 - 2 + 4$$

$$7 \wedge 5$$

$$5 \wedge 3 + 2$$

$$3 - 2/2$$

$$10/2 - 1$$

$$9 * - 2 + 4/2 + 6$$

$$4 + - 2$$

$$3 \wedge 3 \wedge 3 + 1$$

$$3 * 3 * 3 + 2$$

$$4 * 2 + 3 * 2$$

$$6/3/3/1$$

$$6 * 2/2 + 4 * 2 - 3 * 2$$

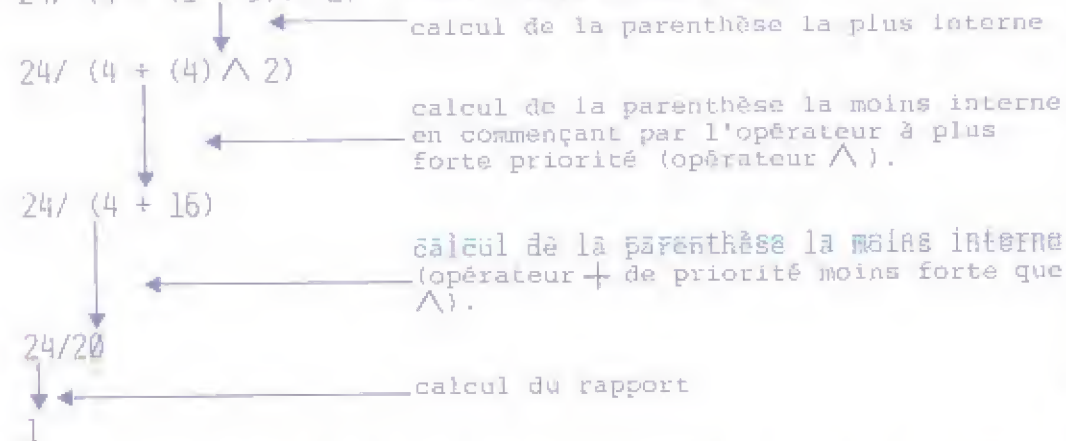
$$40/4 * 6$$

- Il serait, néanmoins, assez fastidieux d'être toujours contraint à une attention soutenue pour éviter des erreurs dans les résultats des opérations multiples. Nous avons donc utilisé des "parenthèses" pour annuler la priorité définie ci-dessus.

Supposons que vous vouliez diviser 24 par $6 + 2$. Vous savez que le fait d'écrire $24/6 + 2$ vous donnera comme résultat 6. Aussi, pour modifier la séquence prioritaire des opérateurs, vous devez écrire $24/(6 + 2)$ où les parenthèses établissent une nouvelle priorité, car ce qui est entre parenthèses est traité en premier.

Dans le cas de parenthèses entre parenthèses, les parenthèses internes sont traitées d'abord.

Exemple: $24/(4 + (1 + 3) \wedge 2)$ sera traité comme suit:



Dans un cas comme $(3 + 3) * (5 * 2)$, où il y a deux groupes de parenthèses mais où aucune n'est incluse dans l'autre, on procède tout simplement de gauche à droite.

Cette expression devient donc: $6 * 10 = 60$

Essayez, comme indiqué plus haut, les quelques exercices suivants:

$$10 \bmod (4 + 1)$$

$$24 / (2 + 2)$$

$$((24/2) + 2)$$

$$3 + (-2 * 2)$$

$$((3 + -2) * 2)$$

$$50/100/(1 * (8 - 4))$$

(Rien ne vous empêche d'en écrire d'autres!!)

EXERCICE III/1

Donnez un exemple pour:

- 1 - Imprimer le résultat d'une opération arithmétique
- 2 - Imprimer une opération arithmétique sans calculer le résultat
- 3 - Imprimer une opération arithmétique et imprimer le résultat à la suite

- 1 - PRINT 3 + 4
- 2 - PRINT "3 + 4"
- 3 - PRINT "3 + 4 = " ; 3 + 4

EXERCICE III/2

Ecrivez en BASIC les expressions suivantes:

$$1 - \frac{nx + m}{ax + b}$$

$$2 - \left. \begin{array}{l} \frac{(3x^2 + y)}{6} + z \\ \frac{(2x^2 - y)}{3} + 5 \end{array} \right\}$$

$$1 - (N * X + M) / (A * X + B)$$

$$2 - ((3 * X \wedge 2 + Y) / 6 + Z) / ((2 * X \wedge 2 - Y) / 3 + 5)$$

Remarque: un bon moyen pour savoir si une expression contient autant de parenthèses ouvrantes "(" que de parenthèses fermantes ")" est d'en faire la somme à partir de la gauche en donnant la valeur +1 à "(" et -1 à ")". A la fin de l'expression le total doit faire 0.

EXERCICE III/3

Donnez le résultat de cette expression en BASIC:

$$(7 * A * B * C) / ((4 * A) * ((D \wedge 2) - A1))$$

avec: $A = 10$

$$B = 2$$

$$C = 20$$

$$D = 10$$

$$A1 = 30$$

VALEUR ABSOLUE :

La qualité principale d'un ordinateur est son extraordinaire rapidité de recherche et de comparaison des données qu'il a en mémoire. Il est donc extrêmement intéressant de lui faire rechercher des valeurs aléatoires. Il existe pour cela la fonction RND.

Essayez par exemple: PRINT RND (9) et... RETURN

et recommencez plusieurs fois, en notant les chiffres qui apparaissent sur l'écran.

Vous constaterez qu'après chaque exécution l'ordinateur affiche sur l'écran un chiffre imprévisible, situé dans une gamme de 0 à 8.

En effet, le chiffre entre parenthèses est la limite que vous fixez à l'ordinateur; il n'en tiendra pas compte pour sa recherche. Si vous vouliez que 9 figure parmi les nombres recherchés, il faudrait écrire RND (10) (chiffres de 0 à 9) ou RND (9) + 1 (chiffres de 1 à 9).

L'opération est identique si vous cherchez des valeurs aléatoires négatives.

Ex: RND (-9) recherchera les valeurs comprises entre 0 et -8

Les applications de cette fonction sont innombrables... (par exemple voyez au chapitre N° V, l'exercice V/4).

VALEURS ALÉATOIRES :

Il est aussi très intéressant de posséder une fonction qui donne la valeur absolue d'un nombre ou d'une expression mathématique:

$(X) = X$ si $X \geq 0$, $(X) = -X$ si $X < 0$.

Votre ordinateur possède cette fonction. C'est la fonction ABS.

- Essayez par exemple:

PRINT ABS (-3) et... RETURN

Le résultat est 3.

- Maintenant essayez

PRINT ABS (3) et... RETURN

Le résultat est 3.

Cette fonction est très utile pour les programmes mathématiques où elle sert couramment.

Elle donne la valeur positive de l'expression entre parenthèses.

SIGNE :

Cette fonction indique le signe d'un nombre ou d'une expression en répondant +1 si le nombre ou l'expression est positif et -1 si négatif. C'est la fonction SGN.

RÉACTIONS POSSIBLES DE L'ORDINATEUR :**CE QUE VOUS DEVEZ FAIRE :**

a) Le message *****SYNTAX ERR** apparaît

Ne pas toucher au réglage de volume mais recommencer l'opération (1).

b) Rien ne se produit

Assurez-vous que le délai de 15 secondes a bien été respecté. S'il n'y a ni >, ni curseur, remettez l'ordinateur en BASIC:

RESET

CTRL

RETURN

B

Mettre le volume un peu plus fort et reprendre à (1).

d) Le message *****MEM FULL ERR** ou **ERR***MEM FULL ERR** apparaît (avec ou sans BIP)

Mettre le volume un peu plus fort et reprendre à (1).

e) L'ordinateur fait BIP et rien n'apparaît

Vous êtes sur la bonne voie! Après le BIP, attendez encore quelques secondes: vous aurez soit un message d'erreur (cas c ou d), ou bien on verra le curseur réapparaître accompagné d'un BIP. Dans ce dernier cas, stoppez le magnéto, puis tapez RUN et appuyez sur RETURN. La bande est "chargée" dans la mémoire de l'ordinateur et le programme commence à se dérouler.

RÈGLAGE DÉFINITIF DU MAGNÉTOPHONE :

Cependant tout le travail que vous avez effectué pour régler le niveau sonore de votre magnétophone n'a pas défini le volume optimum, mais seulement le meilleur réglage en partant du volume le plus faible possible!

Donc, ayant repéré soigneusement ce volume, procédez exactement de la même manière mais en sens inverse, c'est-à-dire en commençant par le volume maximum et diminuant progressivement le niveau jusqu'à obtention du meilleur réglage possible. Le réglage définitif, le meilleur, se trouvera donc à mi-chemin entre celui obtenu à partir du niveau faible et celui obtenu à partir du niveau fort.

NOTEZ-LE TRES SOIGNEUSEMENT, car vous n'aurez (heureusement) plus jamais à recommencer ces opérations.

COMMENT CONSERVER VOS PROGRAMMES SUR CASSETTE :

Dans les chapitres qui vont suivre, vous allez écrire vos premiers programmes.

Si vous désirez les conserver sur cassette, voici comment procéder:

Lorsque votre programme est terminé, branchez le cordon qui relie votre magnétophone à votre ordinateur d'un côté dans la prise cassette OUT de l'ordinateur et de l'autre dans la prise "MICRO EXTERIEUR" du magnétophone.

Faites démarrer votre magnétophone comme si vous vouliez enregistrer.

Tapez `SAVE` puis `RETURN` .

Si tout est bien fait vous entendrez un BIP dans les 15 secondes et le signe `>` réapparaîtra sur l'écran quelques instants plus tard.

Sinon regardez aux: "REACTIONS POSSIBLES DE L'ORDINATEUR" (page précédente).

ATTENTION: si vous avez un réglage de volume à faire, il porte maintenant sur le volume d'enregistrement.

CHAPITRE 4

LA COULEUR C'EST LA VIE (OU LE MODE GRAPHIQUE)

Pour utiliser les couleurs disponibles de votre ordinateur, vous devez passer en mode graphique-mixte!

Pour ce faire, tapez GR. Cette commande (facile à retenir: ce sont les deux premières lettres du mot graphique) efface sur l'écran toute inscription précédente, laissant la place de 4 lignes de texte dans le bas de l'écran.

N.B.: La commande TEXT annule le GR et rend de nouveau l'écran disponible pour écrire sur toute sa surface.

- Donc, vous disposez de 16 couleurs (y compris le noir et le blanc) initialisées de 0 à 15. Vous pouvez les visualiser en chargeant la cassette "Color-Demos" ou plus directement comme nous allons vous l'expliquer plus loin (voir chapitre IX, page 49).

Pour afficher une couleur sur l'écran, vous devez bien sûr spécifier le n° de la couleur choisie et ensuite son emplacement sur l'écran. Pour ce faire, vous devez savoir que l'écran est divisé en 40 colonnes verticales, numérotées de 0 (en haut) à 39 (en bas) et de 40 rangées horizontales numérotées également de 0 (à gauche) à 39 (à droite). Ces "rangées" coupent les "colonnes" créant ainsi une trame de 40 petits rectangles par ligne.

Supposons maintenant que vous désirez afficher sur l'écran un petit rectangle vert pomme. D'abord il faut initialiser la couleur en tapant `COLOR = 10` (10 est le numéro de couleur correspondant au vert pomme).

Ceci signifie que tout ce qui sera placé sur l'écran sera vert pomme jusqu'à ce que vous décidiez de changer de n° de couleur.

Pour placer votre rectangle en haut et à gauche, tapez `PLOT 0, 0` (toujours en haut mais à droite la commande serait `PLOT 39, 0`)

Vous remarquerez qu'il faut toujours indiquer le n° de "colonne" d'abord; ou en termes plus précis: l'axe des abscisses (x) (horizontal) a toujours priorité sur celui des ordonnées (y) (vertical).

Maintenant affichons un rectangle jaune en bas et à gauche:

Tout d'abord: `COLOR = 14` (initialisation nouvelle couleur).

Pour situer l'emplacement tapez: `PLOT 0, 39`

A vous maintenant. Essayez un rectangle violet en bas à droite.

Si vous avez tapé: `COLOR = 5`

`PLOT 39, 39`

Vous avez bien saisi le mécanisme!

Notes

Vous avez noté que les rectangles allumés précédemment en vert pomme, jaune et violet restent affichés sur l'écran. Vous pourriez donc tracer une ligne!

Supposons que vous vouliez tracer une ligne bleu outremer horizontale de la 5ème colonne à la 9ème colonne au niveau vertical 14, vous pouvez évidemment taper:

```
COLOR = 1
PLOT 5,14
PLOT 6,14
PLOT 7,14
PLOT 8,14
PLOT 9,14
```

Heureusement, il y a une manière bien plus simple et plus rapide de procéder!

- Tapez: `HLIN 5,9 AT 14` Par conséquent, pour tracer une ligne horizontale de couleur bleu outremer allant de l'extrême gauche à l'extrême droite et au milieu de l'écran, il est suffisant de taper l'instruction suivante:

```
COLOR = 1
HLIN 0, 39 AT 20
```

De même pour tracer une ligne verticale. Par exemple, `VLIN 10, 19 AT 15` trace un segment de droite vertical entre les lignes 10 et 19, à la colonne 15.

Rappelez-vous qu'en tapant `GR`, vous rendez à l'écran sa pureté originelle. Maintenant vous en savez assez pour essayer vous-même de composer des graphiques simples ou compliqués!

Essayez de tracer une bordure tout le long de l'écran (de couleur vert pomme) puis une croix (de couleur bleu outremer) passant par le centre, et vous pourrez ensuite laisser courir votre imagination...

- Il existe aussi une fonction appelée `SCRN`, dont l'utilité vous apparaîtra immédiatement.

Supposons que vous ayez rempli votre écran de `PLOT`, en couleurs aléatoires, réalisant ainsi un beau patchwork. Essayez donc de déterminer la couleur exacte des coordonnées 17,23 ou 11,13 !! Difficile dans tout ce fouillis, n'est-ce pas? Utilisez donc `PRINT SCRN (17,23)` ou `SCRN (11,13)`.

La réponse sera le n° de la couleur de cet emplacement. Vous pouvez également vous en servir lors d'un test et écrire par exemple:

```
IF SCRN (expr.1, expr.2) THEN expr.
```

(pour l'explication des tests, voir chapitre VI).

EXERCICE IV/1

Tracez un rectangle dont les quatre côtés ont des couleurs différentes. Puis dessinez un point au milieu de l'écran. Ensuite imprimez le chiffre qui correspond à la couleur d'un point de l'écran!

GR

COLOR = 1

HLIN 0, 39 AT 0

COLOR = 2

VLIN 0, 39 AT 39

COLOR = 3

HLIN 0, 39 AT 39

COLOR = 4

VLIN 0, 39 AT 0

COLOR = 5

PLOT 19, 19

PRINT SCRN (0, 1)

NUMÉROTATION DES COULEURS

- 0 : noir
- 1 : bleu outremer
- 2 : vert bouteille
- 3 : bleu océan
- 4 : rouge foncé
- 5 : violet
- 6 : ocre
- 7 : mauve
- 8 : marron
- 9 : bleu clair
- 10 : vert pomme
- 11 : turquoise
- 12 : rouge clair
- 13 : vieux rose
- 14 : jaune
- 15 : blanc

Jusqu'ici, tout ce que nous avons écrit sur l'écran s'inscrivait immédiatement dès que nous avons appuyé sur `RETURN`. C'est naturellement très intéressant mais quelque peu éphémère!

En effet, si par exemple vous aviez tapé `PRINT 3 + 4`, après l'exécution et l'affichage il ne vous était plus possible de retrouver cette instruction; vous étiez obligés, si vous désiriez obtenir à nouveau ce résultat, de taper une nouvelle fois `PRINT 3 + 4`.

Nous allons donc étudier un mode d'exécution différé dont l'utilisation est indispensable pour la construction et le développement d'un programme.

DÉFINITION D'UN PROGRAMME

C'est une série d'instructions dont l'exécution séquentielle permet d'obtenir un ou plusieurs résultats.

Pour signaler à l'ordinateur qu'il s'agit d'un programme et qu'il doit donc stocker les informations sans les exécuter immédiatement, il faut initialiser un début et une fin. Celui-ci comprendra qu'il s'agit d'un début si l'instruction est précédée d'un n° de ligne (comprise entre 0 et 32767) et la fin sera indiquée par la commande `END` tapée à la suite du dernier n° de ligne du programme.

Ex: `10 PRINT 3 + 4` puis frapper la touche `RETURN`
`20 END` puis frapper la touche `RETURN`

Vous remarquerez que, contrairement au mode d'exécution immédiat, rien ne se passe après avoir enfoncé `RETURN`! (seul le signe `>` apparaît). Vous avez compris que l'ordinateur attend une instruction pour l'exécution du programme.

- Celle-ci est `RUN`. Tapez-la sur le clavier (sans n° de ligne cette fois!) puis `RETURN`. Et lisez le résultat.

Tapez encore une fois `RUN` et `RETURN`: le résultat s'affiche de nouveau. Le programme est donc entré dans la mémoire de l'ordinateur.

L'avantage d'un programme, comme nous l'indiquons plus haut, est de pouvoir obtenir autant de fois que nous le désirons le même résultat (cette notion est importante, car elle permet de stocker sur cassette les programmes que vous réaliserez pour que vous puissiez les reproduire au moment où vous en aurez besoin).

Allons plus avant dans l'examen des possibilités d'un programme et la notion de stockage.

Actuellement, votre écran affiche le résultat 7. Mais vous voulez maintenant examiner la liste des instructions de votre programme. Il existe

- une commande toute simple pour cela: `LIST`. Cependant, il est utile d'effacer toute inscription de l'écran.

(Vous vous rappelez comment?)

Appuyez sur **ESC** **SHIFT** **RETURN** puis tapez LIST, puis **RETURN**, et votre programme est affiché sur l'écran. Tapez RUN puis **RETURN** et le résultat apparaîtra de nouveau.

Une seule instruction vous permet de "nettoyer" la mémoire de l'ordinateur des instructions de votre programme: c'est NEW. Tapez-la puis **RETURN**. Maintenant tapez RUN, puis **RETURN** ... Rien, n'est-ce pas?

NUMÉROTATION DES LIGNES

Nous avons vu que nous pouvions utiliser des numéros de 0 à 32767. Il est important de noter que l'ordinateur exécute le programme dans l'ordre croissant des numéros de ligne et pas à pas. Quant au procédé de numérotation, vous pouvez, bien sûr, écrire le programme suivant de cette façon:

```
0 PRINT "K"
1 PRINT "A"
2 PRINT "P"
3 PRINT "A"
4 END
```

Vous avez tapé RUN puis **RETURN** et vous lisez le résultat.

Eh oui! il y a une faute, vous vouliez écrire KAPPA. Que faire dans ce cas?

Il n'y a qu'un seul moyen, retaper la ligne 3:
la ligne 4:
et:

```
3 PRINT "P"
4 PRINT "A"
5 END
```

Alors que si vous aviez tapé ceci:

```
10 PRINT "K"
20 PRINT "A"
30 PRINT "P"
40 PRINT "A"
50 END
```

Il vous était extrêmement facile de rajouter 35 PRINT "P"

En résumé, il est conseillé de garder un écart de 10 numéros entre chaque instruction de manière à intercaler ultérieurement d'autres instructions. D'autant que l'ordinateur reclassera automatiquement les lignes dans l'ordre numérique!

REVENONS SUR LES POSSIBILITÉS DES COMMANDES «LIST» ET «RUN»

LIST: Nous avons vu que si vous tapez LIST, votre programme sera affiché de 0 à N sur l'écran sous conditions, bien sûr, qu'il ne dépasse pas les 24 lignes de capacité de l'écran, sinon il se déroulera jusqu'à la fin affichant ainsi les 24 dernières lignes!)

Mais vous pouvez lui demander l'affichage d'une seule ligne LIST 130 ou d'une partie comprise entre deux numéros LIST 130, 200 où il affichera le programme compris depuis la ligne 130 jusqu'à la ligne 200.

RUN: De la même manière, si vous voulez qu'une partie seulement du programme s'exécute, tapez RUN 130 et l'ordinateur commencera l'exécution à partir de la ligne 130.

Nous avons vu que pour effacer le programme en cours (ce qui est nécessaire pour en inscrire un autre comportant les mêmes numéros) vous devez taper NEW. Cependant, il vous arrivera certainement d'avoir à modifier une ou plusieurs lignes de votre programme.

Pour ce faire, plusieurs possibilités s'offrent à vous.

Vous avez tapé par exemple 130 PRINT 31 + 4 RETURN et vous vouliez, bien sûr, entrer 3 + 4.

Vous pouvez:

- taper 130 puis RETURN, ce qui annulera complètement l'instruction, et retaper correctement la ligne 130, soit utiliser la correction au moyen de ESC
D (voir explications au glossaire).

- ou bien utiliser la commande DEL pour une annulation totale de la ligne.

ex: DEL 130

Mais cette instruction est surtout valable si vous avez, lors d'un long programme, une série de lignes à annuler. Il est alors pratique d'utiliser l'instruction suivante:

DEL 130, 150 qui supprimera les lignes 130 à 150 inclusivement.

UN AUTRE TRUC INTÉRESSANT :

Si, en cours de déroulement de programme, vous désirez l'arrêter à un moment quelconque avant qu'il ne s'arrête de lui-même, il suffit d'appuyer sur les touches CTRL
C et le programme s'arrêtera.

Pour continuer, tapez CON et le déroulement reprendra là où il s'était arrêté.

Le fait de frapper CTRL
C sans appuyer sur RETURN est la seule exception à la règle générale (avec la touche CTRL
X dont nous allons parler

- dans quelques lignes). Ceci nous amène à vous parler d'une erreur relativement fréquente qui consiste à enfoncer la touche RESET au lieu

de **RETURN** (elles sont très proches l'une de l'autre sur le clavier). Ce fait provoquera l'apparition de l'astérisque qui indique que vous êtes passé en langage machine. Rien n'est perdu, il suffit d'appuyer sur **CTRL** et **RETURN**, et vous serez à nouveau en langage BASIC et retrouverez l'intégralité de votre programme.

Attention cependant de ne pas utiliser **CTRL**. Dans ce cas, bien que vous reveniez effectivement en BASIC, vous aurez perdu complètement le programme en cours. A ce propos nous vous rappelons que le **CTRL** n'est utilisé que pour la première mise en route de l'appareil après le **RESET**.

- Nous venons de voir plus haut que pour écrire un programme il était nécessaire de numéroter chaque ligne. Mais si votre programme est assez long, il serait fastidieux de taper à chaque fois les chiffres. Nous avons donc utilisé une astuce pour vous faciliter la tâche.

Supposons que vous vouliez commencer votre programme à la ligne 20. Avant toute chose, tapez **AUTO 20** puis **RETURN**, ce qui aura pour effet d'indiquer à l'ordinateur d'afficher automatiquement une numérotation des lignes de 10 en 10, à partir de la ligne 20. Essayez!

Autre exemple: si vous vouliez démarrer à la ligne 100, et numéroter de 2 en 2, l'instruction serait **AUTO 100,2**.

Même résultat que ci-dessus, mais à partir de la ligne 100, puis 102, 104 etc... (vous pouvez aussi bien entrer **AUTO 500,25** si vous désirez un intervalle de 25).

Quand vous atteignez la dernière instruction de votre programme, vous voyez que sur l'écran est affiché le n° xxx qui attend votre ligne d'instruction suivante. A ce moment, appuyez sur **CTRL** (sans **RETURN**)

ce qui affichera une barre oblique à côté de n° xxx et passera à la ligne (sans numérotation).

Là, il y a deux possibilités: soit vous désirez terminer définitivement votre programme et vous tapez **MAN** (manuel), soit vous pouvez avant cela effectuer un **RUN** ou un **LIST** que vous tapez directement après le **CTRL**.

Dans ce dernier cas, une fois le **RUN** (ou le **LIST**) effectué, vous verrez apparaître le numéro de l'instruction suivante du mode **AUTO**.

- Familiarisez-vous avec la commande **CTRL**, car elle vous permet:

- 1° - de sortir du mode de numérotation automatique,
- 2° - d'annuler en cours de programme une ligne d'instruction erronée, sans sortir de la numérotation **AUTO**,
- 3° - de sortir momentanément du programme **AUTO** pour taper une ligne intermédiaire.

NOTION DE VARIABLE NUMERIQUE, ASSIGNATION

Avant de vous lancer dans les exercices du chapitre V, nous allons vous donner des précisions sur ce que l'on entend par variable numérique et par assignation en informatique.

Lorsque vous écrivez sur papier l'expression:

$$X = 4 + 1/8$$

vous considérez X comme variable car X symbolise une expression numérique en général, à laquelle on assigne (pour le cas présent) la valeur $4 + 1/8$.

Votre ordinateur travaille avec des variables. Si vous écrivez un programme avec une ligne comme ceci:

$$20 \ X = 12 + 15 * 6$$

alors à partir de ligne 20 la variable X aura la valeur 102 pour le reste du programme (à moins que vous ne la modifiez en cours de programme).

Si par exemple vous écrivez (toujours dans le même programme)

$$40 \ A = X/2 - 71$$

alors à la ligne 40 on assigne à A la valeur $X/2 - 71$. Or depuis la ligne 20 X vaut 102; il s'ensuit qu'à la ligne 40 A prendra la valeur $102/2 - 71$, c'est-à-dire -20.

Si vous voulez vérifier essayez le petit programme suivant:

$$20 \ X = 12 + 15 * 6$$

$$40 \ Y = X/2 - 71$$

60 PRINT "X="; X, "Y="; Y

70 END

Les variables peuvent être des lettres, ou des lettres suivies de chiffres ou de lettres.

Exemple:

ALPHA, PR 20B, TEMPERATURE...

- Avant de lancer un programme (avant de faire RUN) les variables utilisées dans le programme ont pour valeur 0.

Si vous désirez remettre à 0 toutes les variables, utilisez la fonction CLR comme dans l'exemple qui suit:

$$10 \ X = 123$$

20 END

RUN

PRINT X (résultat 123)

CLR

PRINT X (résultat 0)

Avant de passer aux exercices, nous allons vous parler d'un type d'assignation très spécial que l'on rencontre constamment en BASIC; par exemple:

$N = N + 2$

Dans ce cas, on retrouve la même variable à droite et à gauche du signe égal.

Mathématiquement, c'est faux. Il suffit de prendre dans l'exemple ci-dessus $N = 0$, on a alors $0 = 0 + 2$, soit $0 = 2$, ce qui est absurde en soi, mais vrai en informatique.

En effet, ici le signe $=$ n'a pas exactement la même signification qu'en mathématiques. En BASIC le signe $=$ signifie: affecter à la variable située à gauche du signe $=$ la valeur calculée dans le nombre de droite.

Prenons un exemple. Veuillez taper le programme suivant:

```
10 V = 0
20 V = V + 1
30 PRINT V
40 V = V + 1
50 PRINT V
60 V = V + 1
70 PRINT V
80 END
```

Maintenant, analysons-le ligne par ligne:

Ligne 10 V prend la valeur 0

Ligne 20 La nouvelle valeur de V est égale à l'ancienne valeur de V ($V = 0$) + 1. Ce qui revient à écrire:
V (nouvelle valeur) = V (ancienne valeur) + 1
soit: V = 0 + 1
d'où: V = 1

Ligne 30 L'ordinateur fait imprimer la valeur de V: V = 1

Ligne 40 Ici, la démarche est la même qu'à la ligne 20:
V = V + 1
soit: V = 1 + 1 D'où: V = 2

Ligne 50 L'ordinateur fait imprimer la valeur de V: V = 2

Ligne 60 $V = V + 1 = 2 + 1$ - Soit: V = 3

Ligne 70 L'ordinateur fait imprimer la valeur de V: V = 3

Nous verrons plus loin des exemples pratiques de ce type spécial d'assignation.

COMMENT MODIFIER UNE LIGNE DE PROGRAMME SANS LA TAPER DE NOUVEAU

Par exemple, supposons que vous écriviez le programme suivant:

```
10 PRINT "CECI EST UN PROGRAMME"
20 END
```

Vous vous êtes aperçu d'une faute d'orthographe à la ligne 10.
Voici la méthode pour corriger sans avoir à retaper toute la ligne.

Faites: LIST 10

Appuyez alternativement sur **ESC** et sur **D**, jusqu'à ce que le curseur soit à la hauteur du numéro de la ligne.

Tapez ensuite alternativement sur **ESC** et sur **A** jusqu'à ce que le curseur soit sur le premier chiffre du numéro de la ligne:

```
10 PRINT "CECI EST UN PROGRAMME"
```

Ensuite, à l'aide de **←** amenez le curseur sur la lettre (E dans le cas présent) que vous désirez changer, et tapez la nouvelle lettre (ici: A).
Puis, toujours à l'aide de **←**, amenez le curseur hors de l'instruction, comme ceci:

```
10 PRINT "CECI EST UN PROGRAMME"
```

Faites **RETURN**, puis tapez LIST: comme vous le voyez, votre instruction est corrigée.

(Il va de soi que ces modifications sont aussi valables pour modifier autre chose que du texte. Par exemple changer le nom des variables, les numéros des lignes, etc...)

D'une manière générale, pour modifier une instruction vous disposez de:

← { qui déplacent le curseur en recopiant dans la mémoire les caractères que ce dernier a "TRAVERSEES"

ESC	A	(déplace le curseur à droite)
ESC	B	(déplace le curseur à gauche)
ESC	C	(déplace le curseur vers le bas)
ESC	D	(déplace le curseur vers le haut)

En effaçant de la mémoire les caractères "TRAVERSEES" par le curseur.

Ces touches vous seront d'une grande utilité pour les instructions longues

Supposons que vous ayez fait le voyage de l'exercice V/2 à trois personnes que votre voiture ait consommé 48 litres d'essence (prix: 2,50 Francs/litre) et que le péage de l'autoroute ait coûté 45 Francs.

Ecrivez un programme qui calcule les dépenses par personne!

```
10 C = 48
20 PE = 250
30 PA = 4500
40 P = 3
50 D = ((C * PE + PA) / P) / 100
60 PRINT "DEPENSES PAR PERSONNES: "; D; "FRANCS"
70 END
```

EXERCICE V/4

Valeurs aléatoires

Utilisez l'ordinateur comme un dé. Ecrivez d'abord un programme qui affiche le résultat d'un coup avec un dé, et ensuite un programme qui donne le résultat d'un coup avec deux dés.

```
10 DE = RND (6) + 1
20 PRINT DE
30 END
```

La première et deuxième ligne peuvent être écrites dans une seule:

```
50 PRINT RND (6) + 1
60 END
```

Maintenant pour 2 dés:

```
80 PRINT RND (6) + RND (6) + 2
90 END
```

(On utilise la fonction RND pour simuler le jet aléatoire des dés).

CHAPITRE 6

MODES DE COMPARAISON (LES TESTS)

Avant de parler des modes de comparaison, nous avons besoin de connaître la notion de branchement inconditionnel.

Par exemple `100 GOTO 20` est un branchement inconditionnel. Cela signifie que la ligne d'instruction n° 100 ordonne à l'ordinateur de retourner continuer son exécution à la ligne d'instruction n° 20.

Exemple: supposons que vous désirez imprimer les uns après les autres les nombres de 1 à 20.

Non! Vous n'allez tout de même pas écrire:

```
PRINT 1
PRINT 2
PRINT 3 ETC...
```

En utilisant l'instruction `GOTO`, vous pouvez écrire:

```
10 N = 1
20 PRINT N
30 N = N + 1
40 GOTO 20
```

- Ce mode d'exécution s'appelle une "boucle ouverte". Ce programme est en quelque sorte "placé sur orbite", étant donné que l'instruction 40 renvoie indéfiniment à la ligne 20. L'ordinateur va ainsi afficher tous les entiers jusqu'à sa capacité maximum.

Cependant, il y a un moyen très simple de lui indiquer une limite précise (par exemple: 10).

Pour cela, nous allons utiliser l'instruction `"IF ... THEN"`

Voici le programme ainsi modifié:

```
10 N = 1
20 PRINT N
30 N = N + 1
40 IF N > 10 THEN END
50 GOTO 20
```

(Remarquez que l'on n'a pas besoin de terminer le programme par `END`, car l'instruction `END` est incluse dans le test).

L'instruction `IF ... THEN` fonctionne de la manière suivante:

Si (`IF`) l'expression mathématique est vraie ($N > 10$), alors (`THEN`) le programme s'arrête (`END`).

Si (`IF`) l'expression est fausse ($N \leq 10$), alors (`THEN`) le programme passe à la ligne suivante (`50 GOTO 20`).

On pourrait également écrire:

```
10 N = 1
20 PRINT N
30 N = N + 1
40 IF N < 11 THEN 20
50 END
```

- Après THEN vous pouvez aussi mettre une assignation ou bien un ordre d'affichage. Exemple:

```
100 IF A = 2 THEN PRINT A
520 IF C = 0 THEN D = 5
```

Appliquons maintenant l'instruction "IF...THEN" et une boucle ouverte au mode graphique.

Entrez le programme suivant:

INSTRUCTIONS

```
NEW
10 GR
20 COLOR = 1
30 COLONNE = 0

40 VLIN 0, 39 AT COLONNE

50 COLONNE = COLONNE + 1
60 IF COLONNE < 40 THEN 40

70 END
```

EXPLICATIONS

Efface le programme précédent

Etablit le mode graphique - mixte

Sélectionne la couleur

Fixe le point de départ sur l'écran (chaque colonne sera remplie successivement de gauche à droite).

Trace une ligne verticale à la colonne 0.

Augmente (à chaque passage) le n° de la colonne de 1.

Colonne, maintenant, égale 1.

Vérifier que la nouvelle valeur de la colonne est inférieure à 40. Si oui le programme retourne à 40 pour placer une nouvelle ligne verticale.

Quand la valeur de la colonne atteint 40, le programme passe à cette ligne et s'arrête.

- Vous constatez que le programme ci-dessus est très clair car en face de chaque instruction figure son explication détaillée. Il serait bien agré-

able de pouvoir l'obtenir sur l'écran, de manière à ce que le débutant sache ce qu'il a fait, sans risque de se tromper sur la signification de sa ligne d'instruction. Cette possibilité existe, c'est la remarque (REM).

Récrivons le programme ci-dessus comme suit:

NEW

5 REM J'ETABLIS LE MODE GRAPHIQUE MIXTE

10 GR

15 REM JE SELECTIONNE LA COULEUR

20 COLOR = 1

25 REM JE FIXE LE POINT DE DEPART SUR L'ECRAN

30 COLONNE = 0

35 REM JE TRACE UNE LIGNE VERTICALE A LA COLONNE 0

40 VLIN 0,39 AT COLONNE

45 REM J'AUGMENTE A CHAQUE PASSAGE LE N° DE LA COLONNE DE 1

50 COLONNE = COLONNE + 1

55 REM VERIFIER QUE NOUVELLE VALEUR DE COLONNE EST < 40, SI OUI PROGRAMME RETOURNE A 40 POUR NOUVELLE LIGNE

60 IF COLONNE < 40 THEN 40

65 REM QUAND VALEUR COLONNE = 40 PROGRAMME PASSE A LIGNE SUIVANTE ET S'ARRETE

70 END

Essayez. C'est plus compréhensible, n'est-ce pas?

Bien entendu, la commande REM n'intervient en aucune façon dans le déroulement du programme et n'affecte pas celui-ci.

Voici un autre exemple, qui utilise la fonction SGN que nous avons déjà vue.

10 I = 0

20 A = I * 2 - 13

30 I = I + 1

40 IF SGN (A) = - 1 THEN 20

50 PRINT "CA Y EST A EST POSITIF"

60 END

EXERCICE VI/1

GOTO
CTRL CON
C

IF ... THEN

Reprenez l'exercice V/4 et changez-le pour qu'il se déroule sans fin.
Arrêtez le programme de temps en temps et faites-le continuer ensuite.

```
10 PRINT RND (6) + RND (6) + 2
20 GOTO 10
```

Changez ce programme de nouveau pour afficher 21 tirages de dés!

```
100 A = 0
110 PRINT RND (6) + RND (6) + 2
120 A = A + 1
130 IF A = 21 THEN 150
140 GOTO 110
150 END
```

EXERCICE VI/2

REM

GR

Ecrivez un programme qui dessine un rectangle au milieu de l'écran (par exemple 9 points sur 6) et mémorisez le titre du programme.

```
5 REM DESSIN D'UN RECTANGLE
```

```
10 Z = 0
```

```
20 GR
```

```
30 COLOR = 12
```

```
40 Y = 17
```

```
50 X1 = 16
```

```
60 X2 = 24
```

```
70 HLIN X1, X2 AT Y
```

```
80 Y = Y + 1
```

```
90 Z = Z + 1
```

```
100 IF Z = 6 THEN 120
```

```
110 GOTO 70
```

```
120 END
```

En modifiant les valeurs pour COLOR, Y, X1 et X2 vous pouvez déplacer le rectangle et changer sa couleur.

CHAPITRE 7

UN PEU DE LOGIQUE

Il est maintenant nécessaire de vous parler de la notion de vérité... que comprend l'ordinateur!

En effet, celui-ci peut distinguer entre ce qui est vrai et ce qui est faux! Etant donné que cette notion a une très grande importance pour la construction de vos programmes, nous allons l'approfondir ci-après.

Il est utile de vous rappeler d'abord les symboles possibles pour des affirmations:

> plus grand que
< moins grand que
= strictement égal à

>= plus grand ou égal à
<= plus petit ou égal à

n'est pas égal à
(ou différent de).

Attention: ne cherchez pas les touches \leq ou \geq , elles n'existent pas. Il faut taper $<=$ ou $>=$ pour obtenir des inégalités larges.

Le symbole $<>$ a la même signification.

Par exemple, l'affirmation $100 > 10$ est évidemment vraie.

Si vous tapez `PRINT 100 > 10`, l'ordinateur répondra 1 qui est le chiffre utilisé pour signifier que l'affirmation est juste.

Dans le cas contraire (exemple: `PRINT 10 > 100`), la réponse sera 0 qui signifie que l'affirmation est fausse.

Les affirmations peuvent inclure aussi bien des variables que des expressions mathématiques ou des nombres. Vous pouvez écrire par exemple:

`JEAN > FREDERIC`

Cette affirmation peut être vraie ou fausse!

En effet, si vous avez pris soin d'affecter auparavant une valeur numérique à chacune de ces variables (par exemple: `JEAN = 4` et `FREDERIC = 8`), alors l'affirmation ci-dessus est fausse et la réponse affichée sera 0.

Par contre, si `JEAN = -4` et `FREDERIC = -8`, l'affirmation est vraie et la réponse sera 1.

- Nous venons de voir que la réponse à vos affirmations est toujours exprimée par 1 ou par 0. De ce fait, on peut les utiliser dans des expressions mathématiques, au lieu d'écrire 1 ou 0.

ex: `PRINT 3 + (4 > 2)` affichera la valeur 4

De même: `A = 5 # 2` affectera la valeur 1 à la variable A

Enfin, une instruction apparemment aussi farfelue que `ALPHA = 18 = 2` signifie simplement que la variable ALPHA sera affectée de la valeur 0 puisque de

toute évidence 18 n'est pas égal à 2!

Cela va encore plus loin!

En effet, si quelque chose n'est pas vrai, c'est faux! et si quelque chose n'est pas faux, c'est vrai! Cette règle, bien particulière aux ordinateurs, ne souffre pas d'exceptions.

ex: tapez PRINT NOT 1 puis PRINT NOT 0 (NOT représente la fonction logique de négation de votre ordinateur).

La réponse sera bien dans le premier cas 0, dans le second cas 1. Evidemment vous pouvez toujours utiliser des expressions telles que PRINT NOT (24 > 2), ce qui donne 0 comme résultat car 24 > 2 est vrai et vaut 1 en logique, et NOT 1 vaut 0.

- Ceci nous amène tout naturellement à un autre système d'affirmations, tout aussi important et que nous appelons "les modes de relation logique".

Ainsi la phrase "un hexagone a 6 côtés" est vraie, ainsi que cette autre phrase "ce manuel est en français".

Relions ces deux phrases comme ceci:

"Un hexagone a 6 côtés AND (et) ce manuel est en français". Bien sûr, l'ensemble de la proposition est vrai!

Maintenant écrivons:

"Un hexagone a 4 côtés AND (et) ce manuel est en français". Bien que seule la deuxième proposition soit vraie, la phrase complète prise dans son ensemble est... fausse! Pour qu'elle soit vraie, il aurait fallu que la 1ère proposition et la 2ème soient vraies toutes les deux.

De même que si vous écriviez:

"Un hexagone a 4 côtés AND (et) ce manuel est en allemand". L'ensemble est faux!

RÈGLE GÉNÉRALE

Quand vous reliez deux phrases ou affirmations par la relation logique AND (et), la réponse sera:

- 1° - l'ensemble est vrai si les deux parties sont vraies
- 2° - l'ensemble est faux si AU MOINS l'une des deux parties est fausse!

Vous voyez sûrement les possibilités que vous offre l'application de cette règle (par exemple si on l'associe à l'instruction IF ... THEN). Mais seule, elle serait insuffisante et elle est complétée par un autre mode de relation logique.

Ecrivons:

Un hexagone a 6 côtés OR (ou) ce manuel est en allemand". Pour que l'ensemble soit vrai, il suffit que l'une ou l'autre des deux propositions soit vraie.

Dans l'exemple ci-dessus, l'ensemble est donc vrai, car un hexagone a bien 6 côtés même si ce manuel n'est pas en allemand!

D'OÙ NOUS TIRONS LA RÈGLE GÉNÉRALE CONCERNANT LA RELATION LOGIQUE "OR":

- a) L'ensemble est vrai si l'une ou l'autre des deux parties est vraie (ou les deux!)
- b) L'ensemble est faux si les deux parties sont fausses.

Voici maintenant quelques exercices à pratiquer sur votre ordinateur, d'abord pour la relation AND, puis avec la relation OR. Mais essayez de trouver la réponse avant qu'elle vous soit donnée par la machine.

```
PRINT 1 AND 1
PRINT 1 AND 0
PRINT 0 AND 1
PRINT 0 AND 0
PRINT (3 > 2) AND 0
PRINT (NOT 0) AND (3 * 3 = 9)
PRINT (4 # 5) AND (4 = 5)
```

```
PRINT 1 OR 1
PRINT 1 OR 0
PRINT 0 OR 1
PRINT 0 OR 0
PRINT (4 # 5) OR (4 = 5)
PRINT 1 OR (0 AND 1)
PRINT ((3 > 4) OR (54 < 327)) AND (NOT 0)
```

- Une autre information est nécessaire:

L'ordinateur considère que 1 est vrai et que 0 est faux. Mais il considère également que tout nombre entier (différent de 0) est vrai. Essayez les deux exemples suivants:

```
PRINT 15 OR 0
PRINT -256 AND 32767
```

Notez bien que la réponse de l'ordinateur pour les relations du type précitées sera toujours 1 ou 0.

Pour ces deux exemples le résultat est 1. En effet, pour le premier, 15 étant différent de 0, il est considéré comme vrai (1); on a donc en fait PRINT 1 OR 0, ce qui est vrai (propriété du OU logique).

Même chose pour le second exemple, car logiquement les deux quantités valent 1, et 1 AND 1 c'est vrai (1).

REMARQUE IMPORTANTE :

Afin de rendre vos instructions parfaitement claires, utilisez les parenthèses. Enfin, voici les nouvelles règles de priorité pour les symboles

étudiées jusqu'à maintenant, par ordre décroissant.

- 1° ()
- 2° NOT, - (signe moins)
- 3° \wedge
- 4° *, /, MOD
- 5° +, -
- 6° $>, <, =, <=, >=, \#$
- 7° AND
- 8° OR

=, <, >

Ecrivez un programme pour 2 variables A et B dont les valeurs sont aléatoires (RND (6)). Affichez le nombre de coups qui sont nécessaires pour arriver à A = B.

```

10  N = 0
20  A = RND (6)
30  B = RND (6)
40  N = N + 1
50  IF A < > B THEN 20
60  PRINT "N = "; N
70  GOTO 10

```

Remarque: ce programme n'a pas besoin de l'instruction END car 70 GOTO 10 fait en sorte que l'ordinateur boucle indéfiniment entre les lignes 10 et 70.

EXERCICE VII/2

AND, OR

Prenez cette fois 4 variables: A et B comme dans l'exercice VII/1, et C1 et C2 avec les valeurs RND (16).

Ecrivez un programme où des lignes verticales seront affichées dans les cas où A = B et c1 = c2.

Changez la valeur de la coordonnée X après chaque ligne (de 0 à 39), ainsi que la couleur.

```

10  GR
20  X = 0
30  A = RND (6)
40  B = RND (6)
50  C1 = RND (16)
60  C2 = RND (16)
70  IF (A = B) AND (C1 = C2) THEN 90
80  GOTO 30
90  COLOR = C1
100 VLIN 0, 39 AT X
110 X = X + 1
120 IF X = 40 THEN END
130 GOTO 30

```

Changez maintenant la ligne 70:

```

70 IF (A = B) OR (C1 = C2) THEN 90

```

CHAPITRE 8

L'ORGANIGRAMME

Lorsque devant votre feuille blanche et votre ordinateur, vous tentez la construction d'un programme, l'approche peut vous en sembler difficile pour peu que de nombreux paramètres entrent en jeu.

Car, malheureusement, l'ordinateur ne peut pas (pas encore!) vous indiquer le meilleur moyen de lui donner des instructions, et l'ordre dans lequel vous allez les introduire n'apparaît pas toujours clairement. Il est donc nécessaire d'utiliser une méthode empreinte de logique et de rigueur pour élaborer l'algorithme qui vous permettra de suivre une logique adaptée à la machine.

Cela est moins difficile qu'il n'y paraît!

Il faut procéder d'abord à l'analyse du problème (que nous désignons par le terme organigramme).

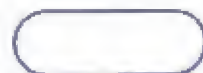
- établir la liste complète des paramètres devant intervenir et leur affecter des variables,
- établir la liste complète des résultats partiels qui seront utilisés en cours de programme et leur affecter également des variables,
- décomposer le problème en actions principales et étudier l'enchaînement entre elles.

Bien entendu, si le programme est complexe, il faut d'abord le détailler en sous-problèmes, sans les analyser à fond pour le moment, mais seulement leur enchaînement, leurs paramètres, et les variables.

Ensuite, reprendre chaque sous-problème et faire d'une manière extrêmement détaillée leur analyse complète.

Nous appellerons la représentation de toutes ces opérations "ORGANIGRAMME"

L'organigramme est une représentation du programme à l'aide des figures suivantes:



Début ou fin



Instructions, initialisations,
opérations diverses



Réservé aux TESTS et généralement à
toutes les instructions qui renvoient
dans des directions différentes.



Flèches de liaison reliant ces figures entre elles (affectées d'un OUI ou d'un NON dans le cas de TESTS).

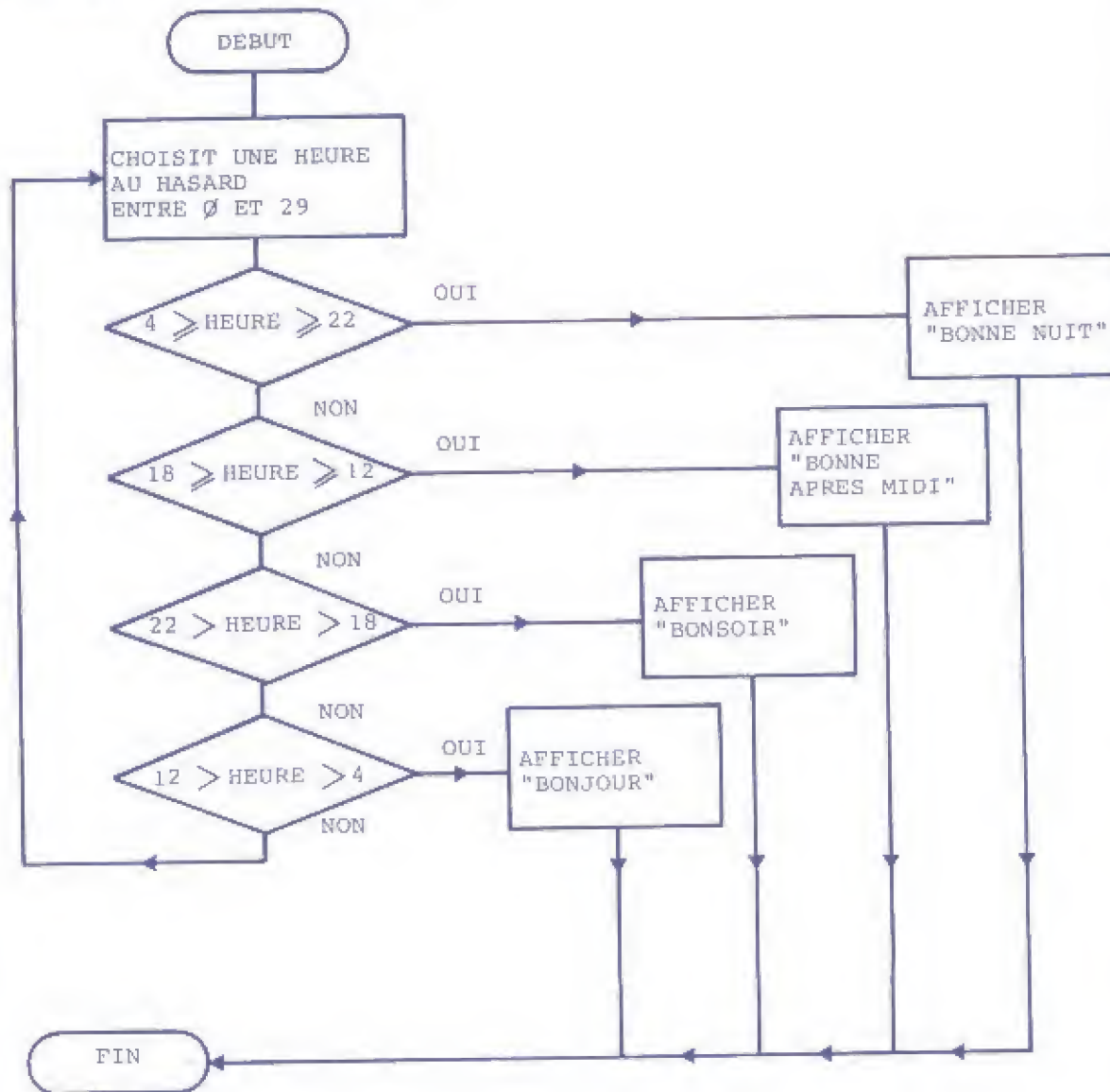


Si l'organigramme déborde le cadre d'une feuille, on utilise le cercle pour indiquer le n° de branchement auquel il se relie.

Voici comme exemple un programme et son organigramme:

```
10  HEURE = RND (30)
20  PRINT HEURE
30  IF (HEURE >= 22) AND (HEURE <= 4) THEN 70
40  IF (HEURE >= 12) AND (HEURE <= 18) THEN 90
50  IF (HEURE >18) AND (HEURE < 22) THEN 110
60  IF (HEURE >4) AND (HEURE<12) THEN 130
65  GOTO 10
70  PRINT "BONNE NUIT"
80  END
90  PRINT "BONNE APRES MIDI"
100 END
110 PRINT "BONSOIR"
120 END
130 PRINT "BONJOUR"
140 END
```

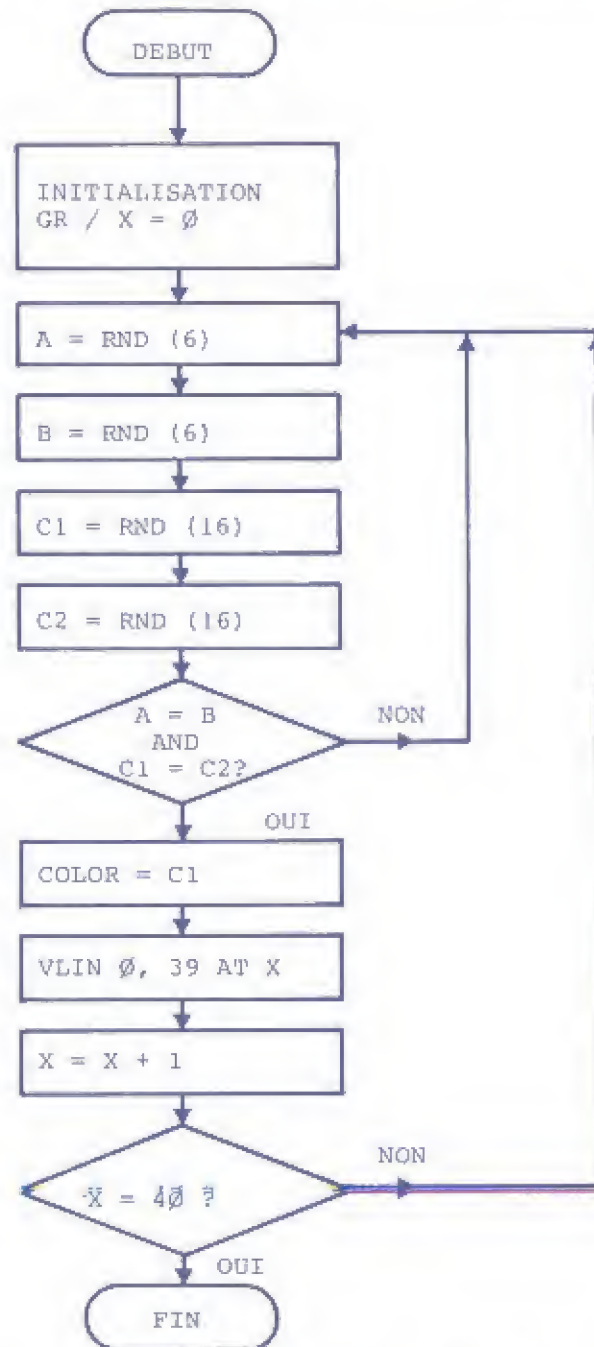
ORGANIGRAMME



A partir de ce chapitre nous vous donnerons les organigrammes des programmes-exemples, et des exercices comportant de nombreux GOTO ou des tests.

EXERCICE VIII/1

Ecrivez un organigramme pour l'exercice VII/2.



CHAPITRE 9

LES BOUCLES (FOR...NEXT...STEP)

Reprenons l'exemple de la boucle ouverte proposée au chapitre précédent:

```
10  N = 1
20  PRINT N
30  N = N + 1
40  IF N < 11 THEN 20
50  END
```

Or, une boucle comporte toujours un haut et un bas, ou un commencement et une fin. Dans le programme ci-dessus, le commencement est la ligne n° 10, la fin la ligne 50. Ce programme affiche les nombres entiers compris entre 1 et 10, 10 étant la limite de la boucle.

- Mais on peut écrire le programme d'une autre manière en utilisant l'instruction FOR ... NEXT :

```
100  FOR N = 1 TO 10
110  PRINT N
120  NEXT N
130  END
```

Il faut utiliser l'instruction RUN 100 pour exécuter ce programme, car si vous tapez seulement RUN, le programme partira de la ligne 10 du précédent. Ou alors tapez NEW avant d'écrire le deuxième programme.

- La ligne 100 est une INSTRUCTION FOR:

Elle initialise la valeur de N à 1, agissant ainsi comme la ligne 10 qui initialise N à 1, ensuite la ligne 110 s'exécute puis la ligne 120 qui est le bas de la boucle.

- L'INSTRUCTION NEXT:

Celle-ci stipule à l'ordinateur d'augmenter de 1 la valeur de la variable N à chaque passage de cette ligne et, tant que cette valeur ne dépasse pas la limite prévue (ici 10), l'exécution continue à la ligne qui suit immédiatement celle de FOR. Dès que la valeur dépasse la limite, le programme ignore la ligne NEXT et passe à la ligne 130 où il s'arrête.

La méthode FOR-NEXT offre donc des avantages indéniables en matière de simplicité, de clarté et de facilité pour la construction d'un programme.

Essayez donc celui-ci:

```
500  GR
510  FOR I = 0 TO 15
```

```
520  COLOR = 1
530  HLIN 0, 39 AT 1
540  NEXT I
550  END
```

Facile n'est-ce pas?

LE STEP

Cette instruction utilisée avec un FOR-NEXT est intéressante à plus d'un titre.

Par exemple:

```
500  GR
505  COLOR = 4
510  FOR I = 0 TO 39 STEP 2
530  HLIN 0, 39 AT I
540  NEXT I
550  END
```

Vous constatez le résultat!

Essayez en modifiant le chiffre qui suit STEP, par exemple 4, 6, 8 etc. La solution est que l'instruction STEP (pas) fait avancer la boucle par "pas" de deux (ou tout autre chiffre jusqu'à 32767). Précédée d'un signe négatif (ex: STEP -2), elle peut même faire exécuter des "pas" en arrière.

Par exemple:

510 FOR I = 39 TO 0 STEP -2 c'est-à-dire de la droite vers la gauche.

N.B.: Par convention, toute boucle est considérée comme avançant par "pas" de 1, le STEP n'a donc une utilité qu'à partir de 2 (sauf dans le cas du pas en arrière où il est nécessaire d'écrire par exemple: FOR I = 39 TO 0 STEP -1).

BOUCLES IMBRIQUÉES :

Il existe une autre possibilité dans l'utilisation de l'instruction FOR-NEXT.

On peut également insérer dans une première boucle de nombreuses autres (en fait jusqu'à 16). La seule règle impérative à respecter est que ces boucles ne doivent en aucun cas se croiser. Cela ne vous paraît pas clair?

Par exemple regardez le programme suivant:

```
10  FOR I = 1 TO 4
20  FOR J = 1 TO 5
30  PRINT I, J
```

```
40 NEXT J
50 NEXT I
60 END
```

En faisant marcher le programme on se rend bien compte de ce qui se passe. Pendant que la variable I a une certaine valeur fixée, la variable J prend toutes les valeurs entre 1 et J.

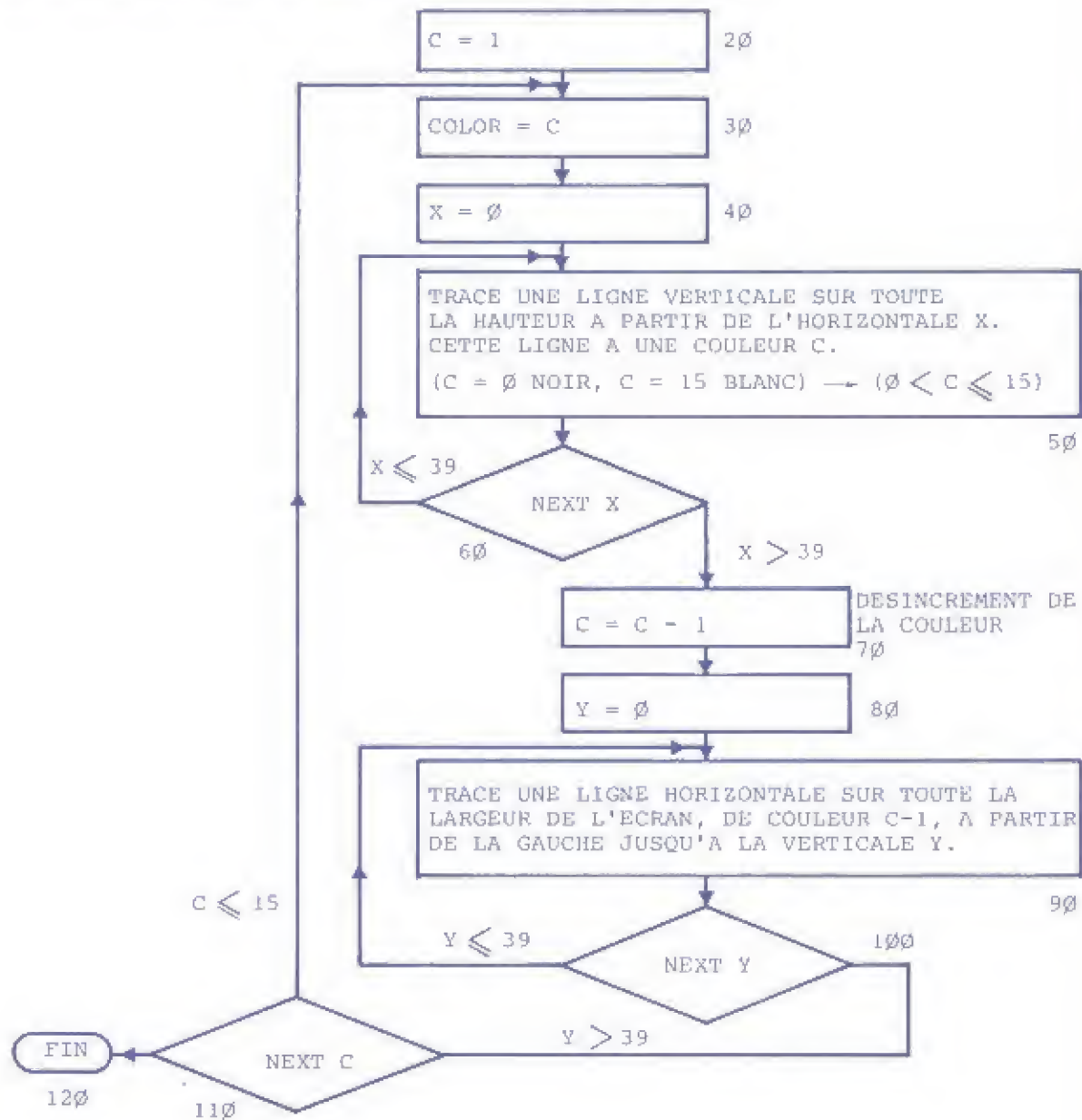
Maintenant étudions les programmes suivants:

PROGRAMME 1 :

```
NEW
10 GR
20 FOR C = 1 TO 15
30 COLOR = C
40 FOR X = 0 TO 39
50 VLIN 0, 39 AT X
60 NEXT X
70 COLOR = C - 1
80 FOR Y = 0 TO 39
90 HLIN 0, 39 AT Y
100 NEXT Y
110 NEXT C
120 END
```


ORGANIGRAMME DU PROGRAMME 1 :

10 GR: initialisation de l'écran, en mode graphique



- ❶ Ceci était un exemple d'une boucle imbriquée à deux niveaux. En voici un autre:

PROGRAMME 2

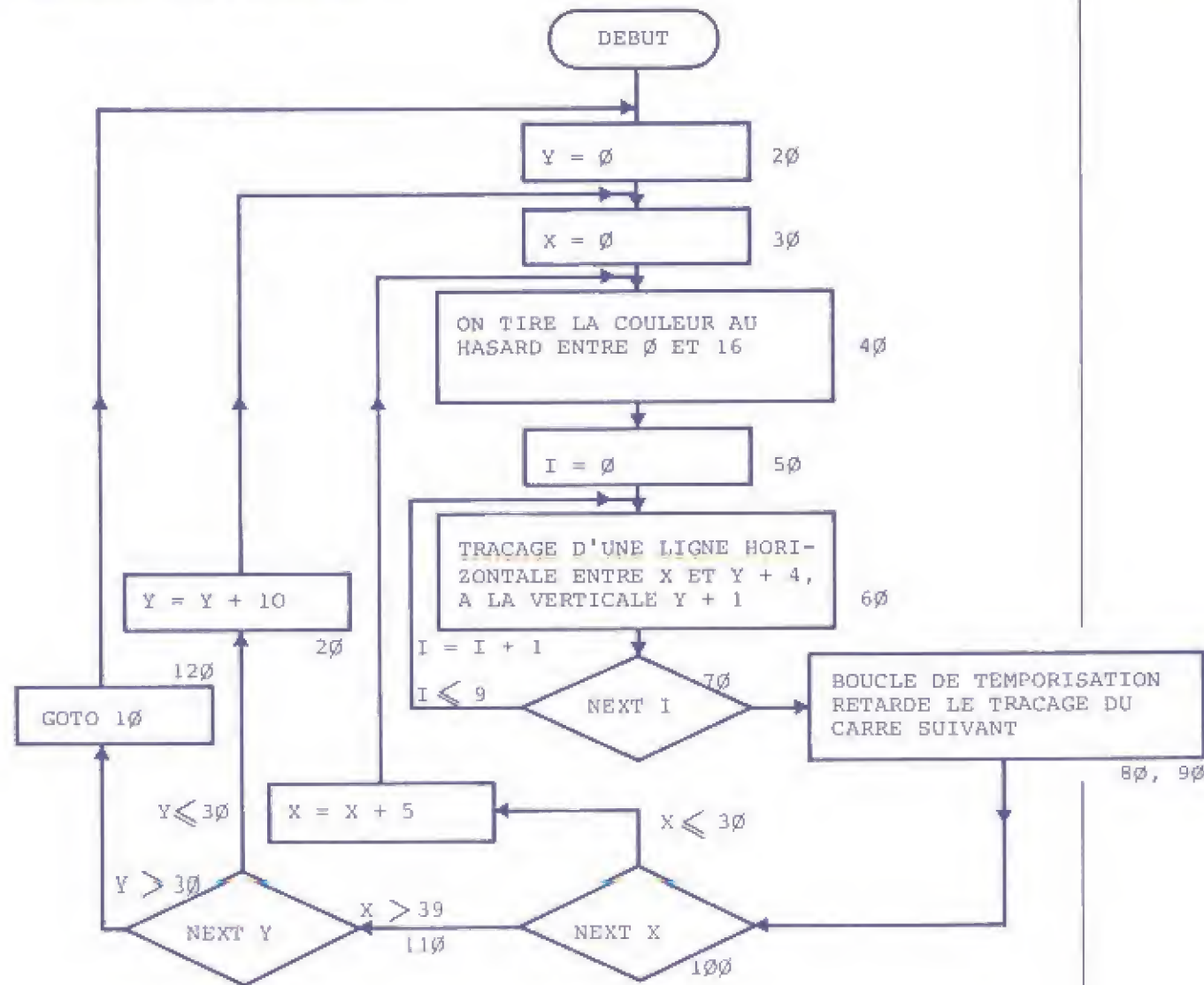
```
NEW
10 GR
20 FOR Y = 0 TO 30 STEP 10
30 FOR X = 0 TO 35 STEP 5
40 COLOR = RND (16)
50 FOR I = 0 TO 9
60 HLN X, X + 4 AT Y + I
70 NEXT I
80 FOR D = 1 TO 100
90 NEXT D
100 NEXT X
110 NEXT Y
120 GOTO 10
```

Ce dernier est à trois niveaux imbriqués. Vous constatez en ligne 80 et 90 la présence d'une boucle dite de "temporisation". Essayez d'en modifier la grandeur (1 TO 250 ou 500 par exemple) et voyez ce qui arrive. Apprenez à bien l'utiliser, car cette boucle spéciale est très intéressante (on voit qu'entre le FOR ... TO ... et le NEXT, il n'y a rien. Cette boucle fait donc travailler l'ordinateur sans rien lui faire faire, ce qui paraîtra comme un temps mort à l'exécution du programme).

Maintenant un exemple de ce qu'il NE FAUT PAS FAIRE!

```
NEW
10 FOR I = 10 TO 20
20 PRINT I
30 FOR J = 30 TO 40
40 PRINT J
50 NEXT I { c'est là où se tient l'erreur! Il aurait fallu
60 NEXT J { placer NEXT I après NEXT J!
70 END
```

L'ordinateur donnera le message d'erreur:***BAD NEXT ERR
STOPPED AT 50



EXERCICE IX/1

FOR ... NEXT ... STEP

Reprenez l'exercice VI/1 et ajoutez une boucle de temporisation, de telle manière qu'au lieu de ne pas attendre entre deux tirages de dés, l'ordinateur laisse un petit laps de temps.

```
1 - 10      PRINT RND (6) + RND (6) + 2
      20      FOR A = 0 TO 300
      30      NEXT A
      40      GOTO 10

2 - 100     PRINT RND (6) + RND (6) + 2
      110     FOR B = 300 TO 0 STEP -1
      120     NEXT B
      130     GOTO 100
```

EXERCICE IX/2

FOR ... NEXT

Construisez trois programmes écrivant les deux phrases:

"MON ORDINATEUR PERD LE NORD"

"MAIS NON... IL M'OBEÏT"

dans des ordres différents:

- 1° - 3 fois la première et 3 fois la seconde.
- 2° - 3 fois (1 fois la première et 3 fois la seconde).
- 3° - 3 fois (1 fois la première et 1 fois la seconde).

```
1° - 10      FOR A = 1 TO 3
      20      PRINT "MON ORDINATEUR PERD LE NORD"
      30      NEXT A
      40      PRINT
      50      FOR B = 1 TO 3
      60      PRINT "MAIS NON... IL M'OBEÏT"
      70      NEXT B
      80      END
```


2° - à changer:

(% = effacer)

```

30    %
75    NEXT A
15    PRINT

```

3° - à changer:

```

50    %
70    %
40    %

```

Tapez un  avant chaque exécution de programmes.**EXERCICE IX/3**

Ecrivez un programme qui compte des pièces de 10 centimes jusqu'à ce que la somme soit de 100 Francs, et qui montre en graphique un tas de pièces de 10 centimes, un tas de pièces de 1 franc et un tas de pièces de 10 Francs.

```

10    GR
20    PRINT
21    PRINT "      10 F          1 F          0.10 F"
22    PRINT
23    PRINT "COMPTEUR DE MONNAIE (DE 0,10 F A 100 F)"
24    A = 0
30    FOR DIX = 1 TO 10
40    FOR UN = 1 TO 10
50    FOR CENT = 1 TO 10
51    COLOR = 1
52    HLIN 33, 35 AT (32 - CENT * 2)
60    FOR N = 1 TO 100
61    NEXT N
62    A = A + 10
70    NEXT CENT
71    FOR N = 1 TO 500
72    NEXT N
80    COLOR = 0

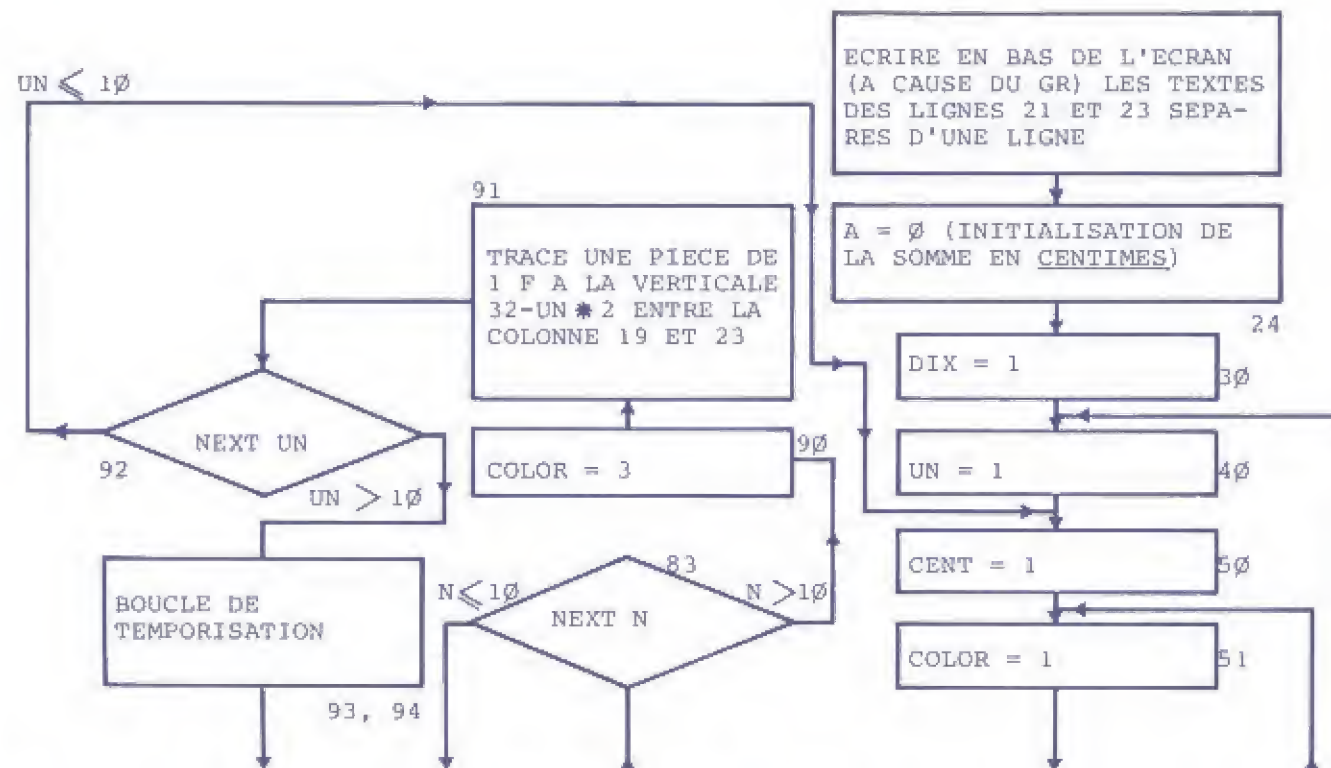
```

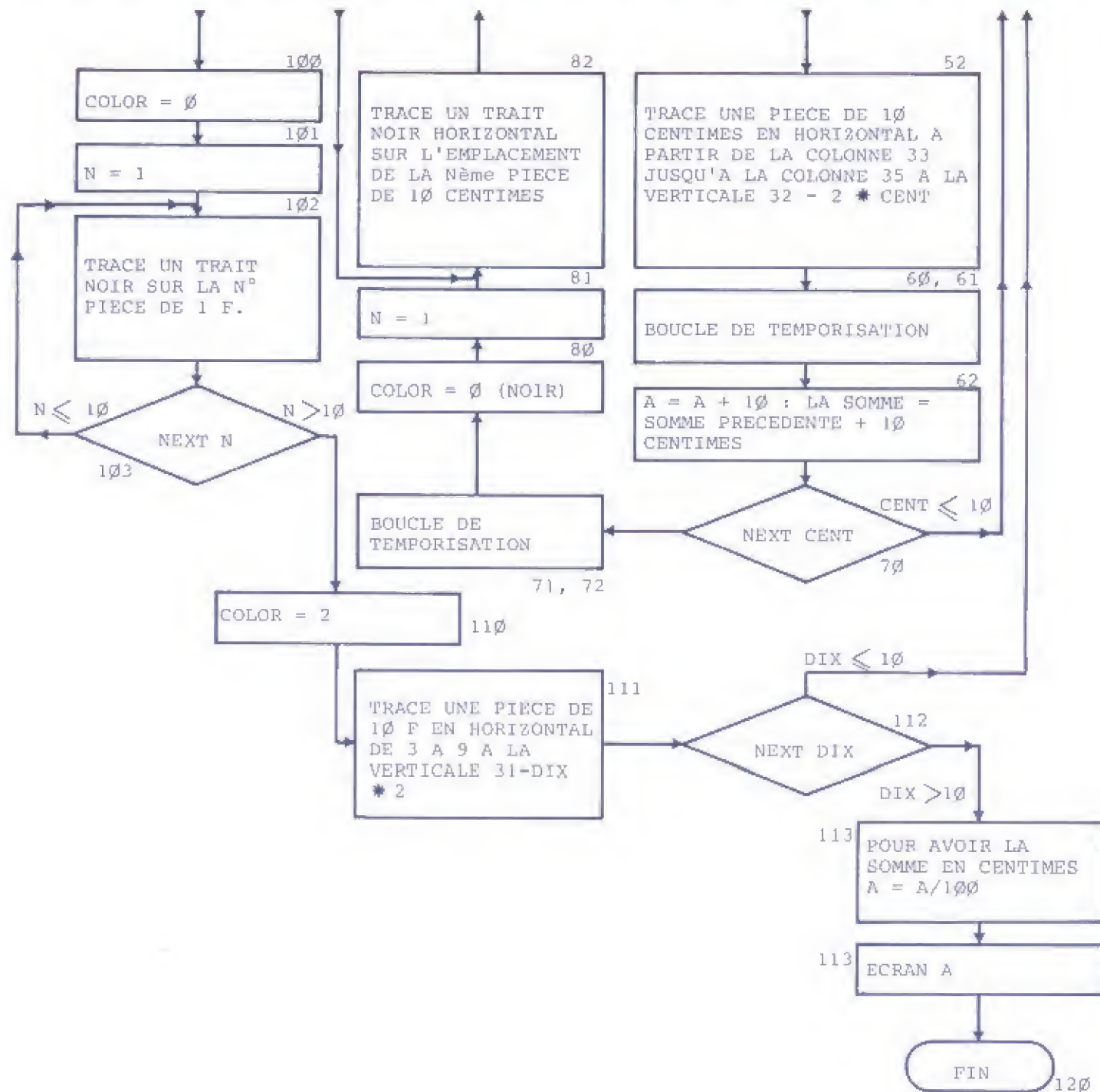
```

81   FOR N = 30 TO 12 STEP -2
82   HLIN 33, 35 AT N
83   NEXT N
90   COLOR = 3
91   HLIN 19, 23 AT (32 - UN * 2)
92   NEXT UN
93   FOR N = 1 TO 500
94   NEXT N
100  COLOR = 0
101  FOR N = 30 TO 12 STEP -2
102  HLIN 19, 23 AT N
103  NEXT N
110  COLOR = 2
111  HLIN 3, 9 AT (32 - DIX * 2)
112  NEXT DIX
113  PRINT "LA SOMME FAIT: "; A/100; "FRANCS"
120  END

```

• (Organigramme de ce programme page suivante)





COMMENT SE PLACER SUR L'ECRAN

(TABULATION, PONCTUATION)

Nous allons reprendre l'une de nos toutes premières démonstrations.

Tapez:

```
NEW
10 PRINT "BONJOUR"
20 GOTO 10
RUN
```

Maintenant tapez le programme suivant:

```
NEW
10 FOR I = 1 TO 10
20 PRINT "BONJOUR",
30 NEXT I
40 FOR I = 1 TO 10
50 PRINT "BONJOUR";
60 NEXT I
70 END
```

Vous voyez qu'en ligne 20 le simple fait de rajouter une virgule après le "BONJOUR" fait que vous obtiendrez plusieurs "BONJOUR" à égale distance les uns des autres. En général le nombre d'espaces laissés entre les textes à afficher est fonction de la longueur du texte entre guillemets. A la ligne 50 le fait de mettre un point virgule après "BONJOUR" génère un nouveau format d'affichage où il n'y a aucun espacement dans le texte à afficher.

Vous pouvez naturellement faire de même avec n'importe quelle inscription...

Essayez par vous-même!

Vous pouvez également utiliser les virgules ou les points-virgules à l'intérieur d'une instruction

Par exemple entrez:

```
NEW
10 JEU = 4
20 AVANTAGE = 1
30 PRINT JEU, AVANTAGE
40 END
```

Et pour améliorer encore la clarté:

```
30 PRINT "LES RAPPORTS JEU ET AVANTAGE SONT " ;JEU, AVANTAGE
```

Ou mieux encore:

```
30 PRINT "LES RAPPORTS JEU ET AVANTAGE SONT " ;JEU;" "; AVANTAGE
```

Mais le plus simple serait:

```
30 PRINT "JEU" ; JEU ; "AVANTAGE" ; AVANTAGE
```

- Récapitulons: - Aucun signe après le texte à afficher → saut de ligne.
 - Une virgule après le texte à afficher → le prochain affichage se fera sur la même ligne mais espacé du dernier.
 - Un point-virgule après le texte à afficher → le prochain affichage se fera sur la même ligne mais collé au dernier.

TABULATION

Supposons que vous vouliez afficher le mot ICI à la dixième colonne (c'est-à-dire au 1er quart de l'écran): Vous pourriez faire:

PRINT" ICI" en laissant donc 10 "blancs" entre le premier guillemet et ICI ... mais l'utilisation de l'instruction TAB est bien préférable.

Tapez par exemple:

```
10 TAB=10
```

```
20 PRINT "ICI" cela aura exactement le même effet!
```

En combinant TAB avec la boucle FOR ... NEXT, vous pouvez programmer des petites choses amusantes.

Par exemple:

```
100 FOR X = 1 TO 24
```

```
110 TAB X
```

```
120 PRINT "#"
```

```
130 NEXT X
```

```
140 END
```

- Vous pouvez également TABuler en vertical, en utilisant l'instruction VTAB

Par exemple:

```
200 FOR X = 1 TO 24
```

```
210 FOR Y = 1 TO X
```

```
220 TAB X
```

```
230 VTAB Y
```

```
250 NEXT Y
260 NEXT X
270 GOTO 200
```

Avant de taper RUN, essayez de vous figurer quel sera le résultat.

- VTAB peut être utilisé pour une action immédiate (sans numérotation de chiffre) alors que TAB doit toujours être utilisé au sein d'un programme. Bien que TAB et VTAB ressemblent quelque peu aux instructions graphiques "PLOT", il existe une différence. En effet, les 40 colonnes disponibles de TAB sont numérotées de 1 à 40, alors que pour PLOT elles le sont de 0 à 39 (la disponibilité est identique, mais c'est seulement une convention). En outre, étant donnée la dimension des caractères, il n'y a que 24 lignes de possibilité d'écriture, d'où les limites de 1 à 24.

Pour l'instruction VTAB, un zéro ou un chiffre dépassant la gamme de TAB ou VTAB donnerait le message ***RANGE ERR

Un nombre plus grand que 255 ou un nombre négatif (-40) donnerait le message ***>255 ERR

La valeur maximale de TAB étant de 255, on peut ainsi l'utiliser au-delà de la largeur de l'écran.

Pour en observer l'effet, essayez le programme suivant:

```
NEW
10 FOR I = 1 TO 255
20 TAB I
30 PRINT I
40 NEXT I
50 END
RUN
```

EXERCICE X/1

Reprenez l'exercice VI/1 et imprimez cette fois les valeurs à la suite, séparées d'un espace,

Puis imprimez les valeurs en forme d'un tableau de 5 colonnes.

```
10 PRINT " " ; RND (6) + RND (6) + 2
20 GOTO 10
```

```
100 PRINT RND (6) + RND (6) + 2,
110 GOTO 100
```

EXERCICE X/2

Imprimez les renseignements suivants sous forme de tableau:

- 1 - Titre: Paris-Pau
- 2 - Départ Paris: 8h00, 18h00, 22h00
- 3 - Arrivée Pau: 14h30, 0h10, 6h30
- 4 - Classe: 1/2, 1/2, 1/2

```
10 VTAB 5
20 TAB 10
30 PRINT "P A R I S - P A U"
40 VTAB 8
50 PRINT "DÉPART PARIS", "ARRIVÉE PAU", "CLASSE"
60 VTAB 10
70 PRINT " 8 H 00" , " 14 H 30" , " 1/2 "
80 PRINT " 18 H 00" , " 0 H 10" , " 1/2 "
90 PRINT " 22 H 00" , " 6 H 30" , " 1/2 "
```

Tapez un  avant chaque exécution du programme!

Si vous désirez calculer la moyenne de plusieurs données, et si elles vous sont connues, vous pouvez entrer le programme suivant:

```
NEW
10 A = 60
20 B = 40
30 C = 50
40 D = 70
50 E = 105
60 X = (A + B + C + D + E)/5
70 PRINT X
80 END
RUN
```

Mais supposons que vous ne connaissiez pas les valeurs des variables au moment de la construction du programme. Vous pouvez alors utiliser une instruction extrêmement précieuse: INPUT.

Exemple:

```
100 INPUT A, B, C, D, E
110 X = (A + B + C + D + E)/5
120 PRINT X
130 END
RUN
```

Vous pouvez également modifier le programme en changeant la ligne 130 en 130 GOTO 100, ce qui vous permettra, sans utiliser la commande RUN à chaque fois, d'entrer autant de valeurs différentes que vous le souhaitez pour les variables considérées.

• Examinons les possibilités de INPUT ...

Vous avez remarqué que les variables entrées après INPUT sont séparées par une virgule. Cela est indispensable si vous désirez entrer plusieurs données. Pour une seule variable, par exemple A, l'instruction serait simplement INPUT A.

Dès que vous aurez tapé RUN, l'écran affichera devant le curseur clignotant un point d'interrogation et l'ordinateur attendra que vous tapiez la ou les données. Attention, vous ne devez entrer que le nombre exact

de données figurant dans le programme.

Exemples:

Si INPUT A, B, C, D, E : entrez 5 données.

Si INPUT X0, ALPHA : entrez 2 données.

Vous pouvez également afficher une explication avant l'introduction des données par le clavier.

Changez ainsi la ligne 100 du programme ci-dessus:

```
100 INPUT "ENTREZ LES DONNEES", A, B, C, D, E
```

Afficher la désignation des variables vous évitera de commettre des erreurs, ce qui peut arriver dans un programme long contenant plusieurs INPUT.

- Il y a une autre façon d'introduire des données en les accompagnant d'un texte indiquant la désignation des variables.

Tapez le programme suivant:

```
10 PRINT "ENTREZ LES DONNEES"  
20 INPUT A, B, C, D  
30 END
```

Dans le cas présent vous voyez que le point d'interrogation se déplace au début de la ligne qui suit celle où le texte "ENTREZ LES DONNEES" est écrit.

Si vous voulez que le point d'interrogation se place sur la même ligne que le texte "ENTREZ LES DONNEES", il suffit pour cela de rajouter une virgule ou bien un point-virgule à la fin de la ligne 10 (après le deuxième guillemet).

Si vous mettez une virgule le point d'interrogation sera espacé du texte.

Si vous mettez un point-virgule le point d'interrogation sera collé au texte.

EXERCICE XI

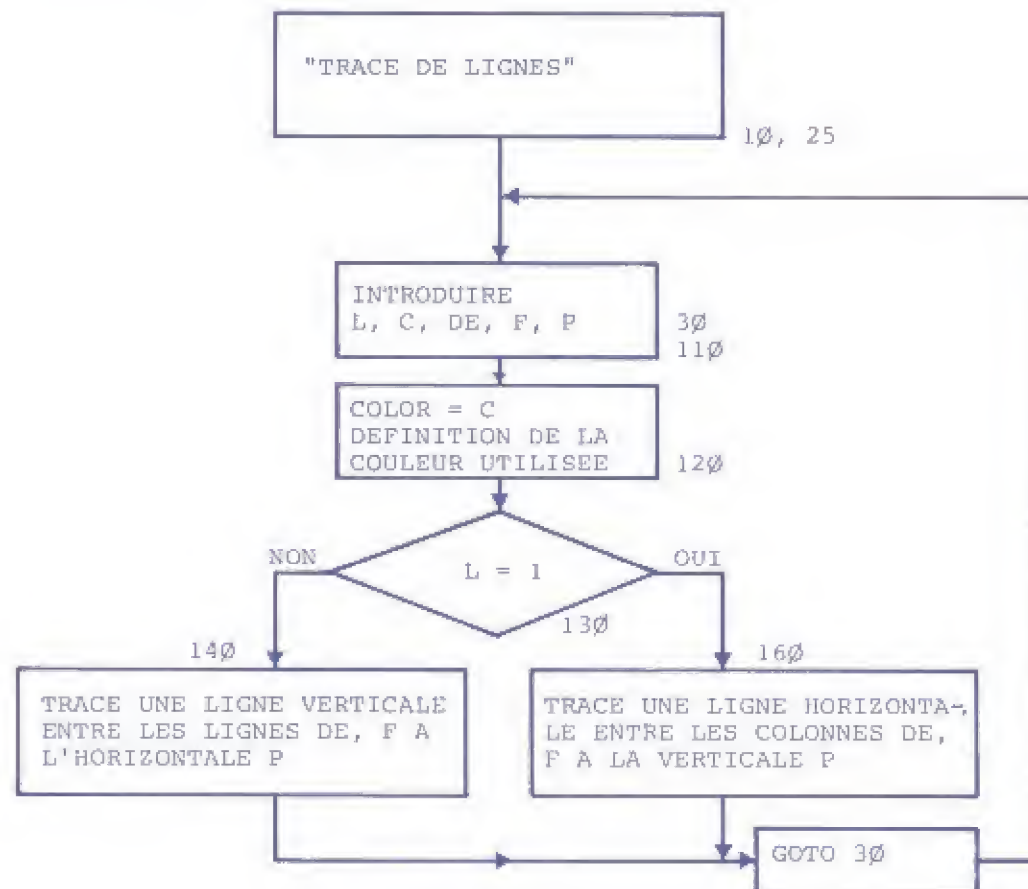
INPUT

Vous auriez pu écrire le programme suivant:

```
10 GR
20 PRINT "TRACE DE LIGNES"
25 PRINT
30 INPUT "ENTREZ 1 POUR UNE LIGNE HORIZONTALE OU 2 POUR UNE LIGNE
40 PRINT                                VERTICALE", L
50 INPUT "ENTREZ LA COULEUR", C
60 PRINT
70 INPUT "POSITION DU DEBUT DE LIGNE", DE
80 PRINT
90 INPUT "POSITION DE FIN DE LIGNE", F
100 PRINT
110 INPUT "POSITION X OU Y", P
120 COLOR = C
130 IF L = 1 THEN 160
140 VLIN DE, F AT P
150 GOTO 30
160 HLIN DE, F AT P
170 GOTO 30
```

Tapez un ESC avant chaque exécution de ce programme.

ORGANIGRAMME DE L'EXERCICE XI



CHAPITRE 12

LES VARIABLES ALPHANUMERIQUES, LEUR DIMENSION

Jusqu'ici, nous nous sommes occupés de graphiques et de chiffres. Mais l'ordinateur peut également traiter les lettres et les symboles. Au lieu de les manipuler un par un, comme les valeurs numériques des variables, l'ordinateur en traite toute une série à la fois. Ceci ne vous étonnera d'ailleurs pas, puisque les mots et les phrases que nous utilisons dans notre langage sont des "séries" de lettres.

Les séries appelées "SERIES ALPHANUMERIQUES" ont des noms qui suivent les mêmes règles que les variables, à ceci près qu'elles sont toujours suivies du signe "\$" comme les quelques exemples ci-après:

NOM\$

A\$

PHRASE\$

Chaque série peut contenir plusieurs caractères. Il faut, avant d'utiliser son nom, indiquer à l'ordinateur le nombre maximum de caractères qu'elle comportera.

- Supposons que vous sachiez que vous aurez besoin d'environ 30 caractères dans une série appelée NOM\$. Vous devez l'indiquer à l'ordinateur par l'instruction DIM (avec ligne numérotée, si celle-ci fait partie d'un programme) et qui s'écrira dans le cas présent 10 DIM NOM\$ (30).

Vous avez maintenant le droit d'écrire:

20 NOM\$ = "UN ROI SANS DIVERTISSEMENT"

Notez que les caractères qui composent cette série doivent être placés entre guillemets. Ce sont ces guillemets, ainsi que le signe \$ à la fin de leur nom, qui permettent de reconnaître aisément les séries alphanumériques.

L'instruction suivante:

30 PRINT NOM\$ affichera le contenu de la série NOM\$ (en l'occurrence, le titre d'un roman de J. Giono). Par ailleurs, si au cours de votre programme vous avez besoin de faire appel plusieurs fois à une chaîne de caractères assez longue, vous pouvez la stocker en mémoire sous la forme de plusieurs variables courtes.

Exemple:

2 DIM A\$ (8)

3 DIM B\$ (10)

4 DIM C\$ (16)

```

10  A$ = "L'ORDINA"
20  B$ = "TEUR EST U"
30  C$ = "N OUTIL PRECIEUX"
40  PRINT A$ ; B$ ; C$
50  END

```

REMARQUE IMPORTANTE :

Il ne peut y avoir plus de 255 caractères dans une chaîne, et comme d'habitude vous ne pouvez entrer qu'une centaine de caractères dans une même instruction.

A ce sujet, attention: tous les blancs d'espace comptent comme des caractères!

Entrez maintenant, en mode direct, les commandes suivantes:

```

DIM MAX$ (255)
MAX$ = "N'IMPORTE QUOI D'UNE CERTAINE LONGUEUR QUE VOUS NE PUISSIEZ
COMPTER D'UN COUP D'OEIL"
Combien de lettres sont stockées dans MAX$? Ne vous donnez pas la peine
de les compter. Il existe une fonction qui vous donnera ce renseignement.
Il s'agit de:

```

● PRINT LEN (MAX\$)

soit en 1'occurrence: 84 (blancs compris, rappelez-vous!)

Pour obtenir une partie d'une chaîne donnée, il suffit d'indiquer les positions numériques du premier et du dernier caractère. Ainsi le segment de MAX\$ compris entre le caractère numéro A et la caractère numéro B s'exprime MAX\$ (A, B).

Par exemple:

```

PRINT NOM$ (13,26) affichera les caractères demandés à savoir ce qui se
tient entre les 13èmes et 26èmes lettres (incluses) de la série (ici:
DIVERTISSEMENTS)

```

S'il s'agit d'un segment dont la fin coïncide avec la fin de la chaîne, on peut omettre la deuxième valeur. Donc tous les caractères à partir du Nème jusqu'à la fin seront représentés par l'expression:

```
MAX$ (N)
```

REMARQUE :

```

Au lieu de taper      10  DIM A$ (10)
                      20  DIM B$ (20)
Vous pouvez écrire:  10  DIM A$ (10) , B$ (20)

```



Voici un programme amusant où la notion de l'utilité des séries alpha-numériques est mise en application:

```
NEW
200 DIM ALPHABET$ (100), X$ (30)
210 ALPHABET$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
220 PRINT "TAPEZ UN CHIFFRE ENTRE A ET 26 ET JE"
222 PRINT "VOUS DIRAIS QUELLE LETTRE SE SITUE A"
224 INPUT "CETTE POSITION DE L'ALPHABET !", P
230 IF P > LEN (ALPHABET$) OR P < 1 THEN 220
240 PRINT ALPHABET$ (P, P) ; "      " ; "EST LA LETTRE N°      " ; P ;
    "DANS L'ALPHABET"
250 PRINT
300 PRINT "TAPEZ UNE LETTRE ET JE VOUS DIRAIS QUEL"
305 INPUT "EST SON N° DANS L'ALPHABET !", X$
310 FOR I = 1 TO LEN (ALPHABET$)
320 IF ALPHABET$ (I, I) = X$ THEN 500
330 NEXT I
340 PRINT "CECI N'EST SUREMENT PAS UNE LETTRE !"
350 GOTO 300
500 PRINT X$ ; "      " ; "EST LE N°      " ; I ; "DANS L'ALPHABET"
510 PRINT
520 GOTO 210
```

Vous pouvez remplacer la ligne 520 par END. Mais ceci vous obligera à taper RUN à chaque fois! Dans le cas présent, pour sortir du programme tapez **CTRL** (vous vous souvenez?).

C

- Il est recommandé de DIMENSIONNER la série avec une valeur de DIM un peu plus élevée qu'il ne paraît nécessaire afin de laisser une certaine marge. En outre, comme dans les lignes 230 et 310, il est préférable d'utiliser la fonction LEN (pour longueur) à la place d'un nombre donné pour que le programme puisse fonctionner si on modifie la série alphanumérique.

Par exemple: si l'on remplace la ligne 210 par: ALPHABET\$ = "0123456789" le programme s'exécutera toujours. Alors que si vous aviez utilisé 26 au lieu de LEN (ALPHABET\$) à la ligne 310, vous obtenez alors le message:

***STRING ERR

Remarquez la manière dont ce programme utilise une boucle pour parcourir la série alphanumérique, une position à la fois, c'est-à-dire en pas à pas. Ce procédé est d'une utilisation facile et très courante.

Ce programme vous montre aussi qu'il est possible de dimensionner plus d'une chaîne à la fois simplement en séparant les éléments par une virgule. Notez aussi les "blancs" entre guillemets (lignes 500 et 240).

D'après vous, quel serait le résultat s'ils étaient omis?

ENCHAÎNEMENT

Il est possible d'ajouter une deuxième chaîne au bout d'une série alphanumérique existante, si la valeur DIM initiale est suffisamment importante pour contenir les deux. Comme nous l'avons vu, l'instruction 100 DIM A\$ (75) alloue jusqu'à 75 caractères à A\$. La longueur réelle de A\$ est LEN (A\$).

Par exemple, si vous tapez:

```
100 DIM A$ (75)
200 A$ = "XYZ"
210 PRINT LEN (A$)
500 END
      RUN
```

le résultat affiché sera 3.

ATTENTION quand votre texte contient des espaces, ceux-ci sont comptés dans la longueur de la chaîne de caractères.

- Pour ajouter un caractère au bout de A\$, tapez:

```
220 A$ (4) = "A"
```

Ajoutez cette instruction au programme ci-dessus, puis entrez:

```
230 PRINT A$, LEN (A$)
      RUN
```

A\$ contient-il bien maintenant XYZA? Ensuite, retapez la ligne 220 comme suit:

```
220 A$ (4) = "ABCDE"
      RUN
```

Le résultat vous surprend-il?

Dans ce système d'enchaînement, $AG(4)$ représente le segment de la série alphanumérique qui commence au 4ème élément. Ainsi, dans la dernière version de ce programme, $AG(4)$ est devenu la lettre A, $AG(5)$ est devenu B et ainsi de suite... Vous venez donc d'enchaîner la série ABCDE à celle de XYZ.

Entrez maintenant ce nouveau programme:

```
NEW
100 DIM UN$ (100), DEUX$ (100)
110 INPUT "TAPEZ LA MOITIE D'UNE PHRASE QUELCONQUE", UN$
120 INPUT "ET ENSUITE LA DEUXIEME MOITIE...", DEUX$
```

Supposons que vous souhaitiez enchaîner $UN\$$ et $DEUX\$$. Vous êtes obligés de connaître la longueur de la première chaîne. C'est pour cela que vous allez utiliser LEN dans l'instruction suivante:

```
130 UN$ (LEN (UN$) + 1) = DEUX$ (remarquez que + est le seul opérateur
                                utilisable avec les variables alphanu-
                                mériques)
```

car vous savez qu'il faudra placer la deuxième série à un élément au-delà de la première. Puisque LEN ($UN\$$) indique la fin de $UN\$$, alors LEN ($UN\$$) + 1 représente l'emplacement du début de $DEUX\$$.

Pour que ce programme s'exécute, tapez:

```
140 PRINT UN$
150 PRINT
160 PRINT
170 GOTO 110
```

et c'est ainsi que l'on pratique... l'enchaînement.

- On peut substituer une série alphanumérique à une autre au moyen d'une instruction telle que:

$X\$ = ALPHABET\$$ où le contenu de ALPHABET\$ est transféré en $X\$$ à condition, bien entendu, que la dimension de $X\$$ soit suffisante. Si ce n'est pas le cas, le message:

STRL OVFL ERR serait affiché, indiquant tout simplement que la longueur de la série alphanumérique a dépassé la DIMension fixée.

N.B.: par convention, un segment partiel d'une chaîne ne peut se placer à gauche d'une instruction de substitution. Ainsi:

$X\$ (3,5) = "XYZ"$ est illégal alors que:

$X\$ = ALPHABET\$ (12,24)$ est parfaitement correct.

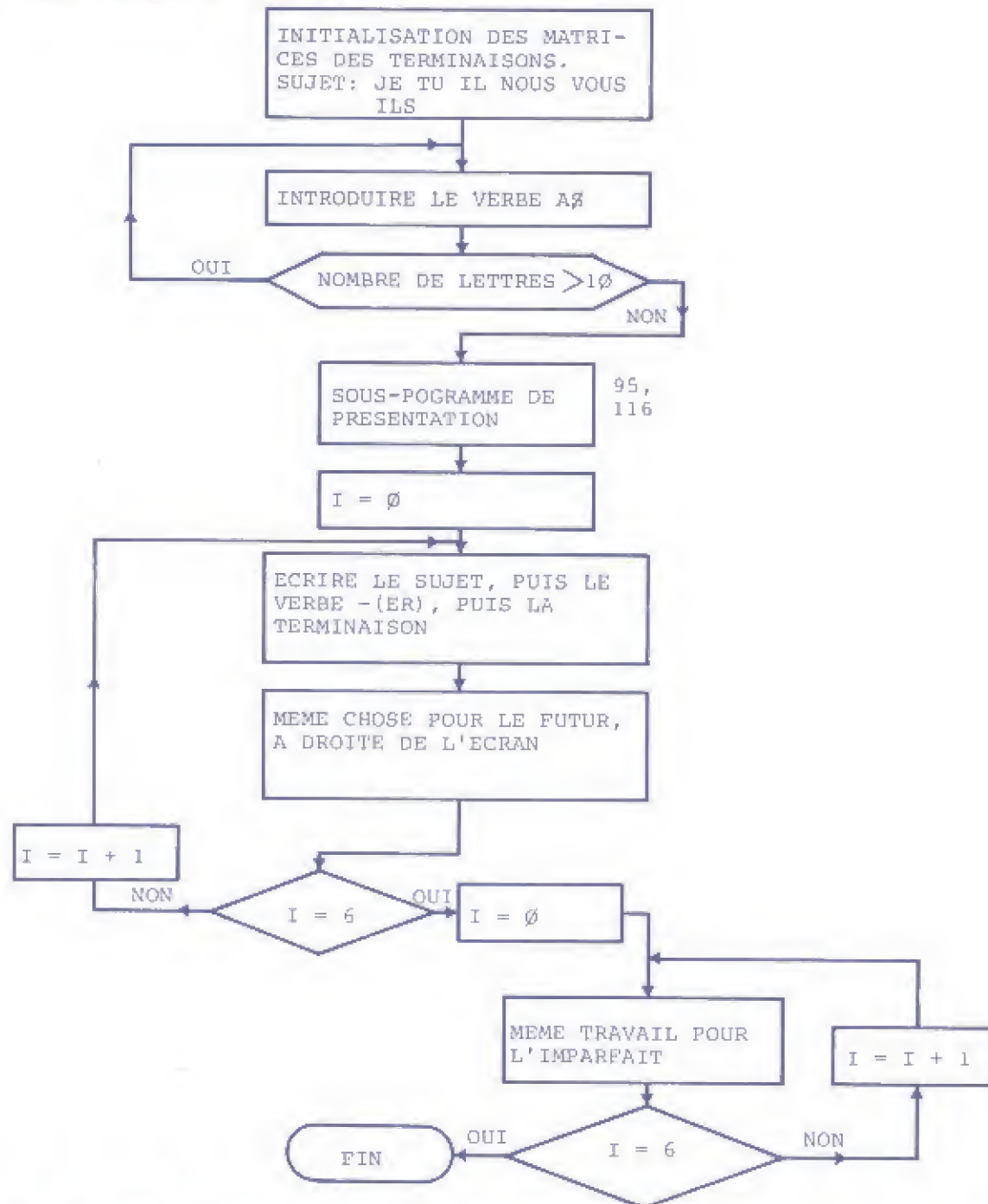
Voici maintenant un programme qui génère des mots aléatoires. La fréquence du choix d'une lettre dépend du nombre de fois qu'elle est inscrite dans ALPHABET\$.

PROGRAMME DE CONJUGAISON DES VERBES DU 1ER GROUPE

```
10 DIM A$(50)
20 DIM SUJET$(24)
25 DIM PRESENT$(18)
30 DIM IMPARFAIT$(30)
40 DIM FUTUR$(30)
50 PRESENT$ = "E ES E ONSEZ ENT"
60 IMPARFAIT$ = "AIS AIS AIT IONS IEZ ALENT"
70 FUTUR$ = "ERAI ERAS ERA ERONS REZ ERONT"
80 PRINT "DONNEZ-MOI UN VERBE DU 1° GROUPE"
81 PRINT "D'UN NOMBRE DE LETTRES INFERIEUR A 10"
85 SUJET$ = "JE TU IL NOUS VOUS ILS "
90 INPUT A$
91 IF LEN (A$) > 10 THEN 80
95 PRINT
96 PRINT "CONJUGAISON DU VERBE" ; A$ (1, LEN (A$))
97 PRINT "....." ;
98 FOR I = 1 TO LEN (A$) - 1
99 PRINT " ";
100 NEXT I
101 PRINT " "
102 FOR I = 1 TO 500
103 NEXT I
105 PRINT
110 PRINT "PRESENT";
111 TAB 20
112 PRINT "FUTUR"
113 PRINT ".....";
114 TAB 20
115 PRINT "....."
116 PRINT
120 FOR I = 0 TO 105
125 PRINT SUJET$ (4 * I + 1, 4 * (I + 1)); " ";
130 PRINT A$ (1, LEN (A$) - 2);
135 PRINT PRESENT$ (3 * I + 1, 3 * (I + 1));
136 TAB 20
```

```
137 PRINT SUJET$ (4 * I + 1, 4 * (I + 1)); " ";
138 PRINT A$ (1, LEN(A$) - 2);
139 PRINT FUTUR$ (5 * I + 1, 5 * (I + 1))
150 NEXT I
155 PRINT
160 PRINT "IMPARFAIT "
170 PRINT "..... "
180 PRINT
190 FOR I = 0 TO 5
200 PRINT SUJET$ (4 * I + 1, 4 * (I + 1)); " ";
210 PRINT A$ (1, LEN (A$) - 2);
220 PRINT IMPARFAIT$ (5 * I + 1, 5 * (I + 1))
230 NEXT I
1000 END
```


ORGANIGRAMME DU PROGRAMME PRÉCÉDENT



EXERCICE XII/1

DIM A\$, B\$

Ecrivez un programme qui obtient deux variables A\$ et B\$:

Faites calculer la longueur de A\$, celle de B\$, et la somme de ces deux longueurs.

Puis imprimez les résultats, et après le contenu de B\$ à la suite du contenu de A\$.

```
10 DIM A$ (100)
20 DIM B$ (50)
30 INPUT "1ER MOT : ", A$
40 INPUT "2EME MOT : " , B$
50 A = LEN (A$)
60 B = LEN (B$)
70 S = A + B
80 PRINT "A$ : "; A; "LETTRES"
90 PRINT "B$ : "; B; "LETTRES"
100 PRINT "A$ + B$ = "; S ; "LETTRES"
110 PRINT
120 A$ (A + 1) = B$
130 PRINT A$
140 END
```

EXERCICE XII/2

A\$ (N, N)

Une personne entre un mot de 5 lettres dans l'ordinateur.

Une deuxième personne essaie de trouver ce mot quelconque de 5 lettres (pas de lettre doublée). L'ordinateur compare le mot à chercher avec l'autre mot. Après il indique le nombre de lettres correctes, puis demande si on a une idée du mot cherché. Sinon, on essaie un autre mot et ainsi de suite, le jeu se termine quand on a trouvé le mot cherché.

Essayez d'écrire un programme qui corresponde à ce jeu!

```
10 DIM A$ (5)
20 DIM B$ (5)
30 INPUT "MOT A CHERCHER", A$
40 END
50 INPUT "O.K. VOTRE MOT", B$
```

```

60 FOR I = 1 TO 4
70 FOR J = 1 + 1 TO 5
80 IF A$(I,I) = A$(J,J) THEN 30
90 IF B$(I,I) = B$(J,J) THEN 50
100 NEXT J
110 NEXT I
120 Z = 0
130 FOR I = 1 TO 5
140 FOR J = 1 TO 5
150 IF A$(I,I) = B$(J,J) THEN Z = Z + 1
160 NEXT J
170 NEXT I
180 PRINT "NOMBRE DE LETTRES CORRECTES";Z
190 IF A$ < B$ THEN 50
200 PRINT
210 PRINT "B R A V O"
220 END

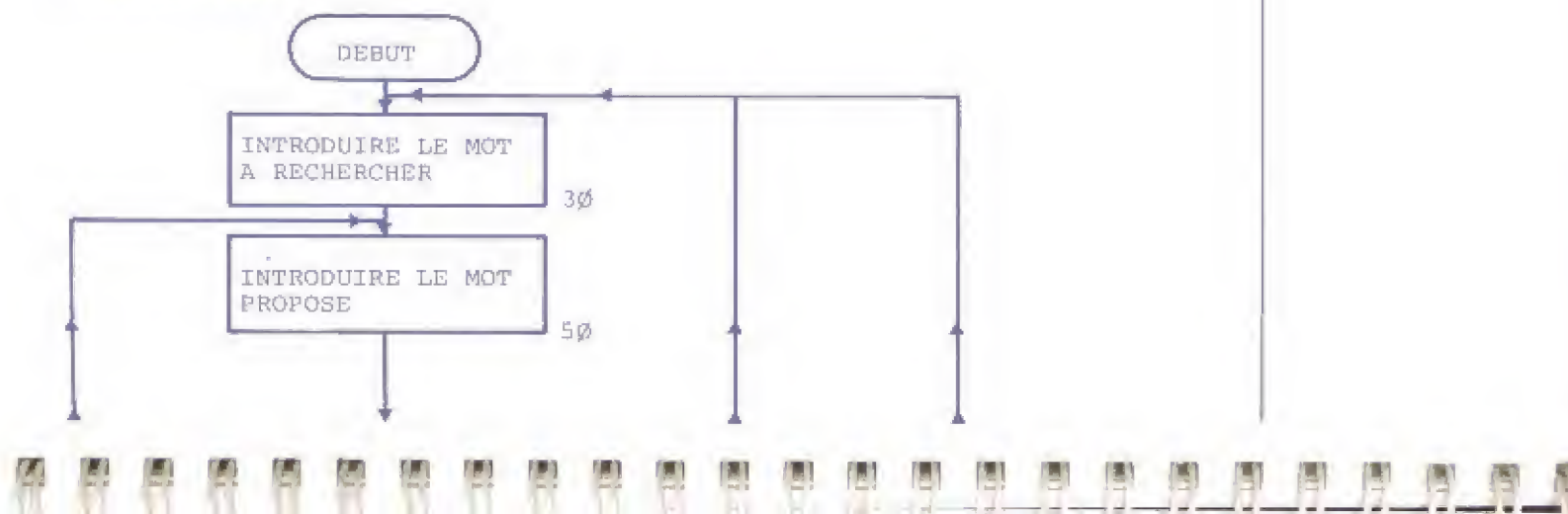
```

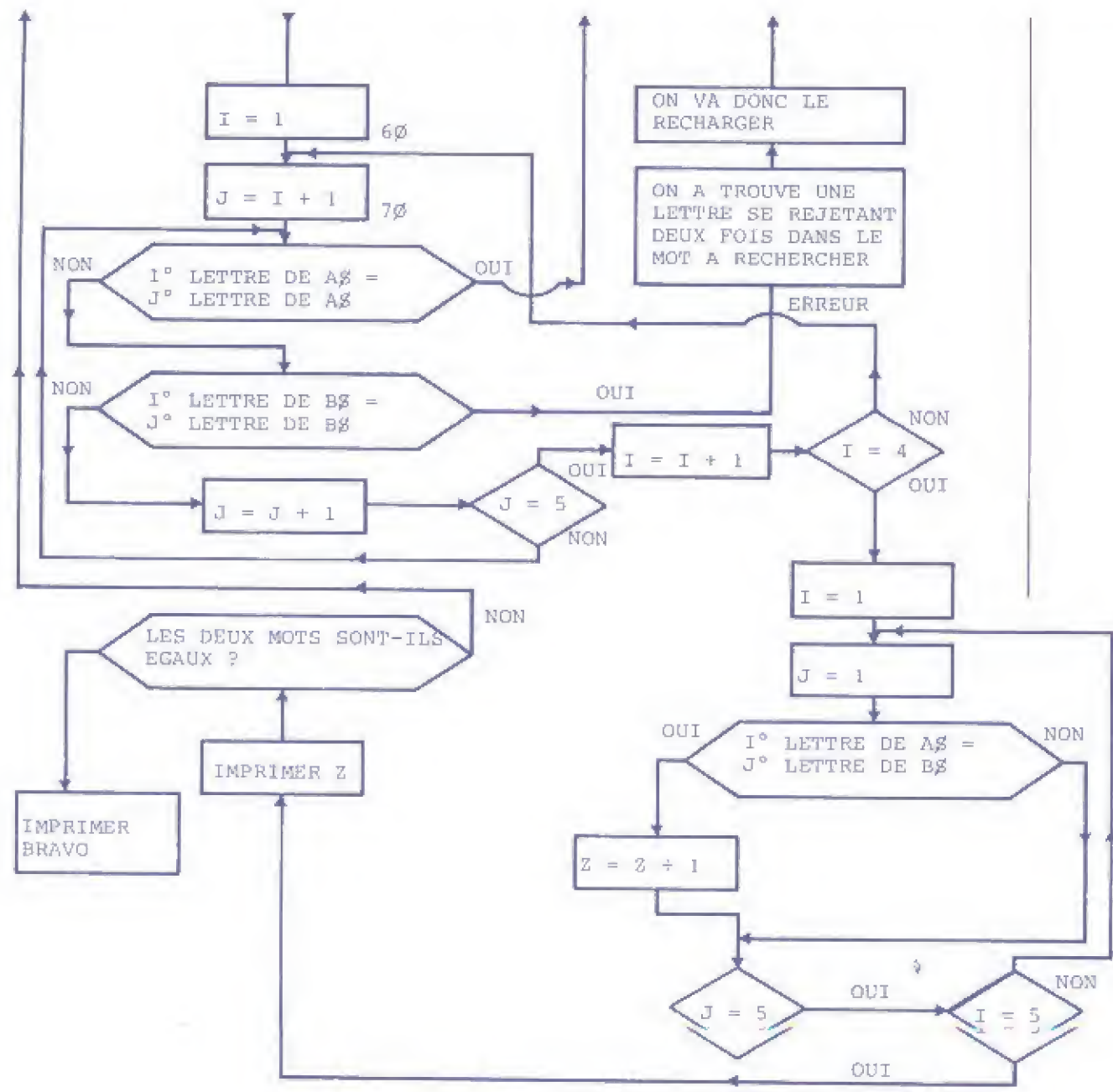
Après avoir effectué le RUN, entrez votre mot.

Il faut que l'écran soit effacé avant que l'on commence à chercher le mot. C'est 40 END qui en donne la possibilité. Appuyez sur **ESC** et **⌫**; après l'effacement on continue avec GOTO 50.

Voir au chapitre XIV pour une autre possibilité (CALL -936).

ORGANIGRAMME





EXERCICE XII/3

Une personne entre une phrase quelconque dans l'ordinateur.

Une deuxième personne essaie de trouver cette phrase. Aide: affichage de la phrase en remplaçant les lettres par des points, avec un espace entre les mots.

Le jeu se termine quand on a trouvé la phrase correcte.

Essayez d'écrire un programme qui corresponde à ce jeu!

```
*10 CALL -936
20 DIM A$ (100), B$ (100)
30 INPUT A$
40 CALL -936
50 A = LEN (A$)
60 FOR B = 1 TO A
70 IF A$ (B, B) = " " THEN 100
80 PRINT " ";
90 GOTO 110
100 PRINT " ";
110 NEXT B
120 TAB 1
130 VTAB 1
140 INPUT B$
150 FOR B = 1 TO A
160 IF A$ (B, B) ≠ B$ (B, B) THEN 180
170 PRINT A$ (B, B) ;
171 GOTO 190
180 PRINT " ";
181 .....
190 NEXT B
200 PRINT
210 TAB 1
220 IF A$ = B$ THEN 240
230 GOTO 140
240 PRINT
250 PRINT "BRAVO - VOUS AVEZ TROUVE"
260 END
```

* Voir chapitre XIV pour explication de cette instruction

CHAPITRE 13

LES REGISTRES, COMMENT LES DIMENSIONNER

LA NOTION DE MÉMOIRE, COMMENT LA TRANSFÉRER

Les registres vous offrent de telles possibilités pour la programmation qu'ils valent largement l'effort qu'il faut consacrer à leur étude.

Ils se distinguent des séries alphanumériques auxquelles, par ailleurs, ils ressemblent beaucoup. Ils sont constitués de chiffres (considérés comme valeurs numériques) et non de lettres et de chiffres (considérés comme des caractères). Pour créer un registre, l'on se sert d'une instruction DIM comme pour les séries alphanumériques.

DIM A (400)

A la différence des séries vues précédemment, il n'y a aucune limite supérieure (à part celle de la capacité de mémoire de votre ordinateur) pour les registres. Il faut simplement se rappeler que plus le programme est long, moins il y a de place mémoire pour les registres et vice versa.

L'instruction DIM ci-dessus, nous a réservé 401 nouvelles variables, qui se comportent exactement comme celles que vous connaissez déjà. Elles sont:

A (0)

A (1)

A (2)

A (3) et ainsi de suite jusqu'à A (400) (prononcez A de 1, A de 2, etc...)

Comme le rôle de ces variables est identique à celui de n'importe quelle autre variable, l'instruction:

A (34) = 45 + A (192) est donc parfaitement correcte.

Voici une application intéressante qui illustre l'emploi des registres. Il s'agit d'un programme qui identifie tous les nombres premiers (c'est-à-dire les nombres qui ne sont divisibles que par 1 et eux-mêmes) à partir de 2 (1, par convention, n'étant pas considéré comme un nombre premier).

Limite de l'ordinateur: 4096 pour 4 K, 16384 pour 16 K, 32767 pour 32 K.

- L'idée générale est la suivante:

Le programme examinera chaque chiffre (appelé EST-CE, puisque'on demande: EST-CE un nombre 1er?) pour voir s'il peut être divisé par un chiffre autre que lui-même. Si oui, EST-CE passera au chiffre suivant, sinon la valeur EST-CE sera imprimée.

Votre micro-ordinateur ITT 2020 dispose d'un moyen excellent pour déterminer si EST-CE est divisible par ESSAI: l'opérateur MOD.

Si EST-CE MOD ESSAI est égal à 0, alors ESSAI est contenu dans EST-CE un nombre entier de fois.

```

NEW
10 PRINT 2
20 FOR ESTCE = 3 TO 4096
30 FOR ESSAI = 2 TO ESTCE - 1
40 IF ESTCE MOD ESSAI = 0 THEN 70
50 NEXT ESSAI
60 PRINT ESTCE
70 NEXT ESTCE
80 END

```

Voici deux questions concernant ce programme auxquelles vous pourrez répondre facilement si vous avez correctement interprété les explications précédentes:

- pourquoi la boucle de la ligne 20 démarre-t-elle au nombre 3 et non au nombre 2?
- pourquoi la boucle de la ligne 30 a-t-elle une limite supérieure à ESTCE - 1?

Cependant, le programme ci-dessus sera terriblement lent! Nous pouvons facilement doubler sa vitesse d'exécution en changeant ainsi la ligne 20:

```
20 FOR ESTCE = 3 TO 4096 STEP 2
```

ce qui permet de "sauter" les nombres pairs qui (à l'exception de 2) ne sont jamais "premiers", puisqu'ils peuvent tous être divisés par 2. La prochaine amélioration se place dans la seconde boucle.

Pour le moment, nous essayons de diviser ESTCE par chaque nombre plus petit que lui-même.

Cela n'est pas nécessaire! En fait, nous n'avons besoin que d'essayer de le diviser par des nombres premiers.

Donc, si nous "conservons" tous les premiers que nous calculons, nous aurons seulement à diviser par ceux-ci et la vitesse d'exécution du programme sera considérablement augmentée.

Et c'est là que nous avons besoin d'un registre (que nous allons appeler PREMIER) pour "conserver" tous les premiers que le programme va trouver.

- Entrez ce nouveau programme:

```

NEW
10 REM PROGRAMME POUR TROUVER DES NOMBRES PREMIERS
20 REM FAISONS DE LA PLACE POUR CES ENTIERS
30 DIM PREMIER (800)
40 REM UTILISONS LE REGISTRE "HAUT" POUR CONSERVER TRACE DU DERNIER
  "PREMIER" TROUVE

```

```

50  HAUT = 1
60  REM PLACONS LE PREMIER NOMBRE "PREMIER"
70  PREMIER (HAUT) = 3
80  REM BOUCLE PRINCIPALE POUR RECHERCHE DES ENTIERS POSSIBLES
90  FOR ESTCE = 3 TO 4096 STEP 2
100 REM RECHERCHER PARMIS LES "PREMIERS" DEJA CONSERVES, ET DIVISEZ-LES
    PAR CHACUN D'ENTRE EUX
110 FOR RESERVE = 1 TO HAUT
120 IF ESTCE MOD PREMIER (RESERVE) = 0 THEN 210
130 NEXT RESERVE
140 REM SI LE PROGRAMME PASSE A CETTE LIGNE, C'EST QUE ESTCE A ETE
    DIVISE PAR TOUS LES PREMIERS ET N'ETAIT PAS DIVISIBLE UNIFORMEMENT
    PAR L'UN QUELCONQUE D'ENTRE EUX
150 REM ESTCE EST PREMIER
160 PRINT ESTCE
170 REM CONSERVEZ CE PREMIER DANS LE REGISTRE
180 HAUT = HAUT + 1
190 PREMIER (HAUT) = ESTCE
200 REM REGARDONS LE PROCHAIN PREMIER
210 NEXT ESTCE
220 END

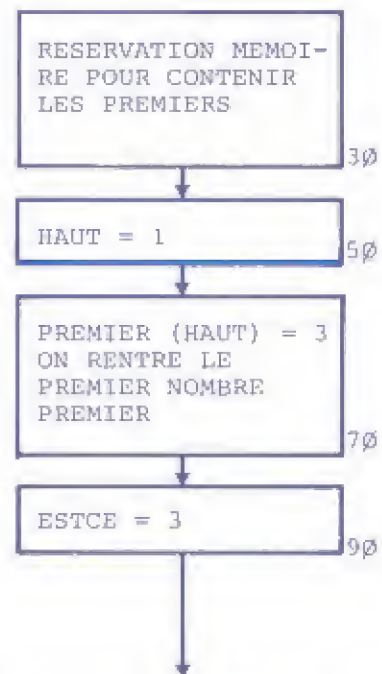
```

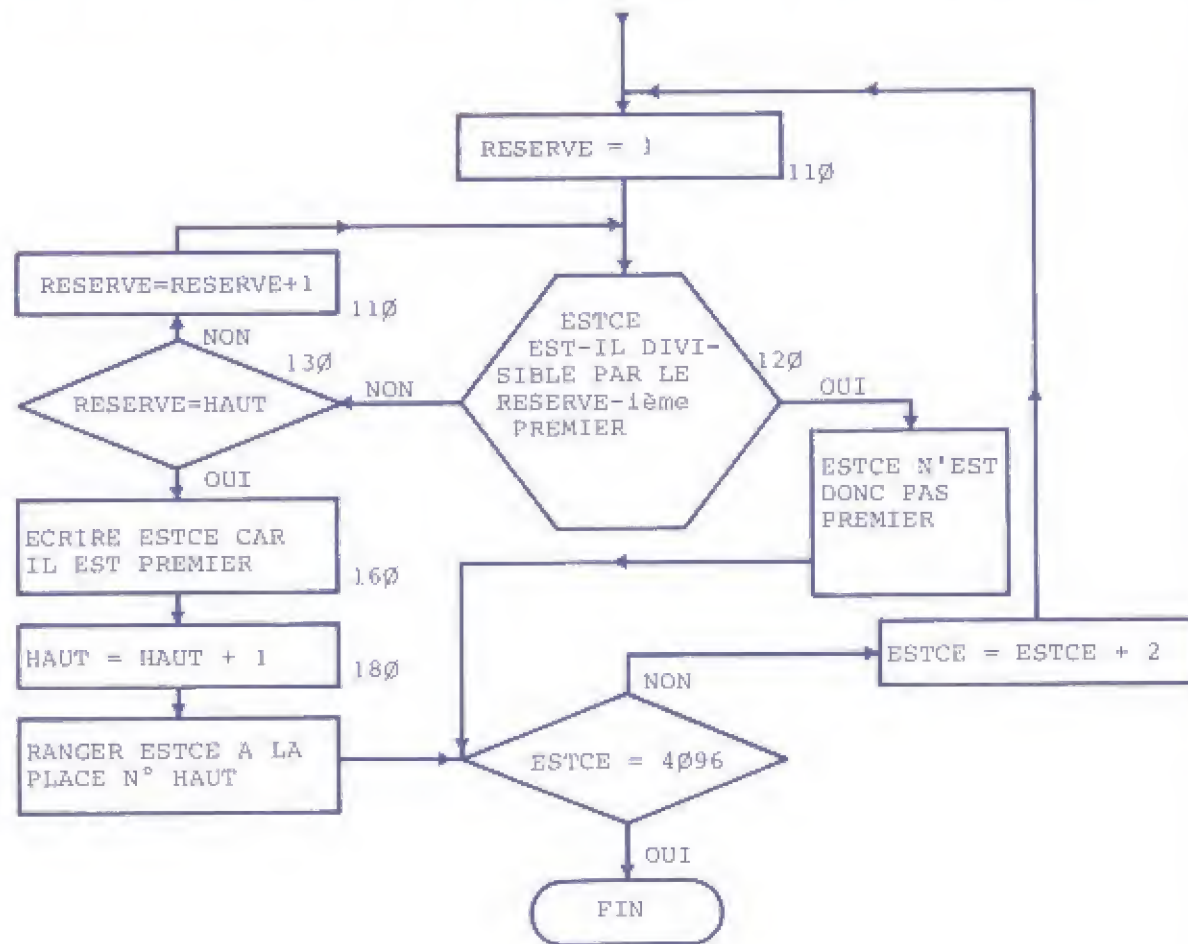
Ce programme imprime donc tous les nombres premiers jusqu'à 4096 excepté 2 et 3. On peut pallier cet inconvénient en ajoutant la ligne

```
5 PRINT 2, 3.
```

N.B.: Ce programme pourrait se dérouler encore plus rapidement si vous divisiez ESTCE, non pas par chaque nombre entier, mais seulement par ceux inférieurs à la racine carrée de ESTCE.

Pensez un peu à cette solution, et modifiez vous-même le programme en conséquence.





MESSAGES D'ERREURS À PROPOS DES REGISTRES :

Si un des nombres "conservés" en registre est négatif ou plus grand que celui indiqué en DIM, le message ***RANGE ERR apparaîtra.

Mais il est toujours facile de savoir pourquoi et où l'erreur s'est produite. En effet, l'ordinateur peut dire: ***RANGE ERR

STOPPED AT 190

d'où vous tapez LIST 190 et il vous imprimera:

190 PREMIER (HAUT) = ESTCE

et ainsi, vous saurez que la variable HAUT est sortie de la gamme. De plus, pour voir ce qui est arrivé tapez:

PRINT HAUT

et constatez exactement quelle valeur a stoppé le programme.

Jusqu'à maintenant, les quelques programmes que nous avons entrés n'ont pas toujours eu un déroulement parfait, notamment si notre construction n'était pas convenable!

Il existe un moyen d'en suivre le déroulement pas à pas afin de constater à quel endroit exact notre programme a divergé. Nous appellerons cela le "dépiage".

Néanmoins, afin de le démontrer, nous allons construire un programme de recherche des racines carrées d'un nombre quelconque.

D'abord quelques données de bases:

Si vous multipliez un nombre par lui-même, par exemple $5 * 5$, vous obtenez le carré de ce nombre, soit 25. Inversement, nous disons que 5 est la racine carrée de 25. Beaucoup de nombres entiers, par exemple 20, n'ont pas pour racine carrée un nombre entier. En effet, 5 est trop grand, et 4 (dont le carré est 16) trop petit. Il est donc clair que la racine carrée de 20 est entre 4 et 5.

Le problème est donc d'écrire un programme qui recherchera et trouvera la racine carrée approximative d'un nombre. D'abord, remarquez que le plus grand nombre que votre ITT 2020 puisse contenir étant 32767, sa racine carrée est 181. De ce fait, la gamme des racines carrées que ITT 2020 peut traiter se trouve obligatoirement entre 0 et 181.

Construisons le programme:

Nous savons que la racine carrée d'un nombre (que nous appellerons NOMBRE) doit être comprise entre 0 et 181. Nous allons donc initialiser deux compteurs MIN (pour minimum) et MAX (pour maximum).

10 MIN = 0

20 MAX = 181

Nous allons maintenant demander la recherche d'un nombre compris quelque part entre MIN et MAX et nous regarderons si ce nombre (que nous allons appeler QUETE) est trop grand ou trop petit, étant donné qu'un bon moyen de se "promener" entre MIN et MAX est de choisir un nombre situé à mi-chemin des deux extrêmes. La ligne suivante s'écrira:

```
30 QUETE = (MIN + MAX)/2
```

Maintenant, nous avons besoin de savoir si QUETE est plus grand ou plus petit que la valeur exacte de la racine carrée recherchée. Cette valeur exacte, si elle est portée au carré, sera égale à NOMBRE. Portons donc au carré notre QUETE:

```
40 CARRE = QUETE ^ QUETE
```

Ainsi, nous pouvons comparer son carré à NOMBRE!

- Les instructions ci-dessus comprennent l'essentiel du programme. En effet, si CARRE est trop grand, alors nous savons que la véritable racine carrée est plus petite que QUETE, d'où nous abaisserons la valeur de MAX vers QUETE. Par contre, si CARRE est trop petit, alors nous savons que la véritable racine est supérieure à QUETE, et nous augmenterons la valeur de MIN vers QUETE.

Aussi longtemps que MAX et MIN ne sont pas égaux, nous devons continuer à examiner les nouvelles valeurs de QUETE. Les valeurs les plus sûres étant entre les nouvelles valeurs de MIN et MAX.

Nous allons ajouter cette ligne:

```
80 GOTO 30
```

qui indiquera à l'ordinateur de chercher à nouveau.

De cette manière, nous rapprochons de plus en plus l'une de l'autre, les valeurs de MIN et MAX, jusqu'à ce que la racine carrée approximative soit contenue entre elles.

```
50 IF CARRE > NOMBRE THEN MAX = QUETE
```

```
60 IF CARRE < NOMBRE THEN MIN = QUETE
```

Quand MAX et MIN sont égaux, nous avons trouvé la racine carrée, donc nous écrivons:

```
70 IF MAX = MIN THEN 90
```

et à la ligne 90, ceci:

```
90 PRINT "LA RACINE CARREE DE" ; NOMBRE
```

```
110 PRINT "EST APPROXIMATIVEMENT" ; QUETE
```

Voici le programme complet:

```
10 MIN = 0
```

```
20 MAX = 181
```

```
30 QUETE = (MIN + MAX)/2
```

```
40 CARRE = QUETE ^ QUETE
```

```
50 IF CARRE > NOMBRE THEN MAX = QUETE
```

```
60 IF CARRE < NOMBRE THEN MIN = QUETE
```

```
70 IF MAX = MIN THEN 90
```

```
80 GOTO 30
```



```
90 PRINT "LA RACINE CARREE DE" ; NOMBRE
100 PRINT "EST APPROXIMATIVEMENT" ; QUETE
```

Pour essayer le programme écrivons:

```
5 INPUT "QUEL EST LE NOMBRE?" ,NOMBRE
```

Tapez RUN et la question? : entrez 34 RETURN

- Oui... apparemment rien ne se produit, le programme fonctionne mais ne répond pas! (on dit que le programme boucle). C'est là que nous allons chercher pourquoi à l'aide du mode de dépiage! Stoppez avec CTRL.

C

Entrez (sans numéro de ligne) PRINT NOMBRE, QUETE, MIN, MAX, CARRE afin de voir ce que contenaient ces variables quand vous avez stoppé le programme. Ceci vous donnera une idée précise de ce qui se passe. NOMBRE, bien sûr, avait la valeur 34 et vous pouvez vous demander pourquoi nous nous occupons de cette valeur, puisque le programme indique clairement qu'elle ne doit pas changer.

Effectivement, NOMBRE n'est pas supposé changer mais en l'affichant vous savez avec certitude que sa valeur n'a pas été modifiée. En effet, il peut y avoir une très grande différence entre ce que le programme est supposé faire et ce qu'il fait réellement!

C'est un principe qu'il ne faut jamais oublier: ne tenez rien pour vrai, avant que vous ne l'ayez constaté! Donc, QUETE est 5, MIN est 5 et tout cela est parfaitement correct, cependant MAX affiche 6!! Que s'est-il passé?

- ... "DEPISTAGE"

Avant que nous allions plus loin, essayons de voir ce qui s'est passé pendant le déroulement du programme:

tapez: 1 DSP MIN

2 DSP MAX

ce qui signifie que vous demandez "l'affichage" des valeurs de MIN et MAX. Vous pourriez, bien sûr, ne pas numérotar les lignes, mais alors vous ne pourriez plus utiliser RUN car cette commande détruit les fonctions DSP. (Vous pourriez, néanmoins, démarrer le programme par un: GOTO 5, ce qui équivaut, bien sûr, à un RUN, à la différence que GOTO ne remet pas les variables à 0).

Donc, le programme va vous demander à nouveau une valeur pour CARRE, et vous entrez à nouveau le nombre 34. Attendez avant de presser RETURN! En effet, vous allez devoir vous tenir prêt à presser CTRL instantané-

C

ment pour stopper le programme. Si vous avez été assez rapide, votre écran devrait afficher:

```
# 10 MIN = 0  
# 20 MAX = 181  
# 50 MAX = 90  
# 50 MAX = 45  
# 50 MAX = 22  
# 50 MAX = 11  
# 60 MIN = 5  
# 50 MAX = 8  
# 50 MAX = 6  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5  
# 60 MIN = 5
```

STOPPED AT 70

Cet affichage montre qu'aux lignes 10 et 20 les valeurs respectives de MIN et MAX étaient bien conformes à leur initialisation, soit 0 et 181. Ensuite, vous pouvez voir que les valeurs de MAX décroissent jusqu'à 11 tandis que la valeur de MIN (qui n'est pas affichée) reste à 0. Cette démonstration vous explique parfaitement la fonction dsp. Chaque fois qu'une instruction contenant un INPUT ou un =, et contenant la variable affectée d'un DSP, est lue, l'écran affiche le n° de ligne, le nom de la variable, et sa valeur. Pratique non...? (Pour annuler la fonction DSP à l'intérieur d'un programme, écrire (n° de ligne) NODSP).

REMARQUE: Si vous avez en programme la variable MAX affectée d'un DSP, ainsi qu'une instruction du type MAX = MAX -2, la fonction DSP concerne le MAX à gauche du signe =.

Mais continuons l'examen de notre affichage.

Depuis sa valeur 11, MAX atteint 8 puis 6 et rien ne sera changé après cette dernière valeur, tandis que la valeur de MIN qui est de 5 depuis la ligne 60, continuera d'afficher cette valeur. Reprenons l'exemple de $5 * 5 = 25$. Le CARRE est donc 25 puisque QUETE est à 5. Donc, pour le nombre 34, CARRE sera toujours inférieur à NOMBRE, MIN sera toujours rapporté à QUETE, et MAX jamais égal à MIN.
Changez le programme avec cette instruction:

```

70 IF MAX = MIN + 1 THEN 90

```

Tapez RUN pour plusieurs valeurs de NOMBRE, cela vous donnera toujours une réponse avec une erreur maximum de 1. Bien sûr, cela ne vous donnera pas toujours le plus proche entier de la racine exacte, mais l'écart ne sera jamais supérieur de 1 à la réponse exacte.

- Mais... ce programme ne vous donnera pas de réponse, si vous entrez un nombre qui est un carré parfait (comme 25). Essayez de trouver un moyen à l'aide de DSP et... d'un peu de réflexion!

Vous n'avez pas trouvé? Alors changez la ligne 50 comme suit:

```
50 IF CARRE >= NOMBRE THEN MAX = QUETE
```

Le fait d'affecter la valeur de MAX quand CARRE est soit égal, soit supérieur à NOMBRE, permet au programme de donner également les racines des carrés parfaits, sauf si QUETE peut ou ne peut pas être la racine carrée de NOMBRE. Cependant, MAX l'est toujours; donc, nous changeons la ligne 100:

```
PRINT "EST, ARRONDI AU PLUS PROCHE ENTIER SI NECESSAIRE," ; MAX
```

Maintenant, le programme est parfaitement au point et donnera la réponse exacte pour un carré parfait, ainsi que la réponse arrondie au plus proche entier, si CARRE n'est pas un carré parfait.

Vous vous souvenez que nous avons donné précédemment un exemple d'utilisation des registres pour la recherche de nombres premiers, et nous vous avons suggéré en N.B. d'utiliser les racines carrées pour accélérer cette recherche. Si vous ne l'avez déjà trouvé, nous allons établir un programme utilisant ces deux procédés.

Jusqu'ici donc, quand nous recherchions les premiers, nous divisions par chaque premier inférieur au nombre testé. En réalité, il n'est pas nécessaire de diviser par chaque premier, mais seulement par chaque premier inférieur ou égal à la racine carrée du nombre testé.

Ainsi, en combinant les deux procédés, nous pouvons augmenter considérablement la vitesse d'exécution, surtout si l'on prend soin de placer le plus d'instructions possibles sur la même ligne.

- Ce que nous allons voir maintenant.

```
NEW
10 DIM PREMIER (800)
20 PREMIER (1) = 2
30 PREMIER (2) = 3
40 PRINT 2, 3.
50 HAUT = 2
60 FOR QUETE = 3 TO 4096 STEP 2 : J = 2
70 IF QUETE MOD PREMIER (J) = 0 THEN 100 : IF PREMIER (J) >= MAX THEN
80 : J = J + 1 : GOTO 70
80 PRINT QUETE, : HAUT = HAUT + 1 : PREMIER (HAUT) = QUETE : MIN = 0 :
MAX = 181
90 RACINE = (MAX + MIN)/2 : ESSAI = RACINE * RACINE : IF ESSAI >= QUETE
THEN MAX = RACINE : IF ESSAI < QUETE THEN MIN = RACINE : IF MAX =
```



```

      MIN + 1 THEN 100 : GOTO 90
100  NEXT QUETE
110  END

```

INSTRUCTIONS MULTIPLES SUR UNE SEULE LIGNE

Vous venez de voir dans le programme précédent que sous le même n° de ligne, l'on pouvait écrire toute une série d'instructions différentes sous la condition de les séparer d'un ":" cette commande remplaçant donc un n° de ligne.

Ce procédé extrêmement courant:

- 1 - se caractérise par une plus grande rapidité d'exécution,
- 2 - occupe moins de place en mémoire,
- 3 - permet, si le programme est relativement court, d'obtenir son affichage complet sur la surface disponible de l'écran.

Par contre, la lisibilité et la compréhension ne sont pas facilitées. Ainsi nous vous conseillons de n'utiliser ce procédé qu'à partir du moment où vous serez familiarisés avec la programmation.

- Il existe également une autre fonction de "dépistage", elle aussi très utile, que nous pouvons utiliser quand un programme présente quelque anomalie. Elle permet de suivre le déroulement ligne par ligne. Il s'agit de la commande TRACE. Dans un programme, le texte qui s'affiche correspond à ce que vous avez demandé au programme d'afficher. La fonction TRACE fait afficher en plus du texte du programme tous les numéros des instructions du programme au moment de leur exécution.

Cela est bien pratique lorsque vous utilisez de nombreux GOTO ou GOSUB (GOSUB = commande pour sous-programmes, voir chapitre suivant).

```

10  FOR I=1 TO 10
20  NEXT I
40  PRINT "BONJOUR"
100 END

```

Faites RUN: le programme écrit: "BONJOUR"

Maintenant faites TRACE, puis RUN. Sur l'écran vous voyez s'afficher:

```

# 10 # 20 # 10 # 20 # 10 # 20 # 10 # 20 # 10 # 20
# 10 # 20 # 10 # 20 # 10 # 20 # 10 # 20 # 10 # 20
# 40 BONJOUR
# 100

```

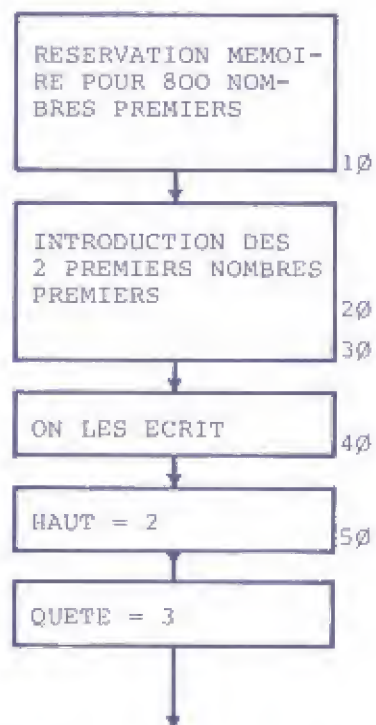
Vous avez vu: l'écran suit le programme.

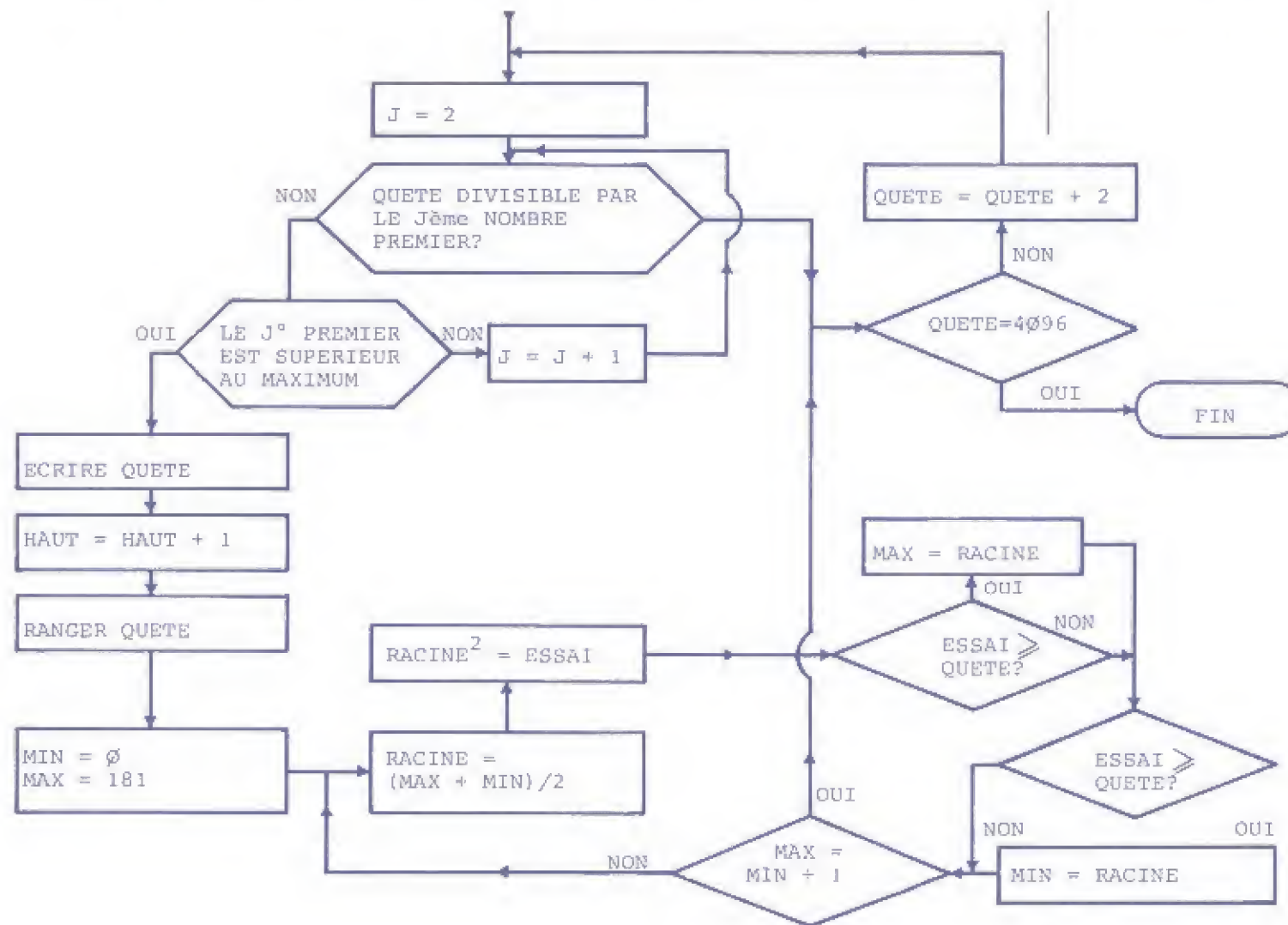
Dans ce cas, vous pouvez inscrire 5 TRACE. Et si vous voulez n'examiner le déroulement que jusqu'à la ligne 40, vous écrirez 45 NOTRACE.

De toute façon, vous serez obligé d'écrire NOTRACE, sinon vous auriez toujours en mémoire la commande TRACE.

A la différence de DSP qui est annulé par un RUN, TRACE n'en est pas affecté et peut être utilisé en mode d'affichage direct, donc sans numérotation. N'oubliez pas, même dans ce cas, de notifier NOTRACE quand vous n'en aurez plus besoin.

ORGANIGRAMME "NOMBRES PREMIERS" 2





EXERCICE XIII/1

X (1), X (2), ...

Ecrivez un programme qui utilise trois registres pour mémoriser les coordonnées x et y et la couleur de différents points (max. 20 points).

Après avoir mémorisé les points, affichez-les sur l'écran.

```

10 DIM X (20)
20 DIM Y (20)
30 DIM C (20)
40 INPUT "NOMBRE DE POINTS", Z
50 FOR N = 1 TO Z
60 INPUT "X, Y (DE 0 A 39) : ", X(N), Y(N)
70 INPUT "COULEUR (DE 0 A 15) : ", C(N)
80 NEXT N
90 GR
100 FOR N = 1 TO Z
110 COLOR = C(N)
120 PLOT X(N), Y(N)
130 NEXT N
140 GOTO 40

```

EXERCICE XIII/2VOITURES D'OCCASION

<u>VOITURE *</u>	<u>ANNEE</u>	<u>COULEUR *</u>	<u>PRIX</u>	<u>VENDU</u>
2CV	1975	VERT	10000	NON
GOLF	1975	BLEU	15500	NON

Ecrivez un programme qui affiche un tableau comme ci-dessus pour 12 voitures d'occasion.

Changez au fur et à mesure l'indication NON en OUI pendant l'exécution du programme!

```

10 REM ENTREE DES DONNEES
20 DIM VOIS (48), A (12), C (60), P (12), V (36), OUI (40)
30 INPUT "NOMBRE DE VOITURE (DE 1 A 12) ?", Z
40 FOR N = 0 TO Z - 1
50 INPUT "VOITURE (EN 4 LETTRES) : ", VOIS (N * 4 + 1)

```

```

60 INPUT "ANNEE:", A (N + 1)
70 INPUT "COULEUR (EN 5 LETTRES): ", C$ (N * 5 + 1)
80 INPUT "PRIX (LIMITE A 32767): ", P (N + 1)
90 V$ (N * 3 + 1) = "NON"
100 NEXT N
110 REM AFFICHAGE EN TABLEAU
120 PRINT "          VOITURES D'OCCASION"
130 PRINT : PRINT
140 PRINT "VOITURE", "ANNEE", "COULEUR", "PRIX", "VENDU"
160 PRINT
170 FOR N = 0 TO Z - 1
180 PRINT VOIS (N * 4 + 1, N * 4 + 4), A (N + 1), C$ (N * 5 + 1, N * 5 + 5), P (N + 1), V$ (N * 3 + 1, N * 5 + 3)
190 NEXT N : PRINT
200 INPUT "VENTE D'UNE VOITURE? (OUI/NON)", OUI$
210 IF OUI$ = "OUI" THEN 240
220 INPUT "VOULEZ-VOUS RECOMMENCER? (OUI/NON)", OUI$
230 IF OUI$ = "OUI" THEN 30 : GOTO 200
240 INPUT "NUMERO DE LA RANGEE?", X
250 IF X = Z THEN 280
260 L = LEN (V$)
270 OUI$ (4) = V$ (X * 3 + 1, L)
280 V$ ((X - 1) * 3 + 1) = OUI$
290 GOTO 170

```

* Exemple: - GOLF

```

      2CV (+ 1 BLANC) = 4
- BLEU (+ 1 BLANC)
- VERT (+ 1 BLANC) = 5

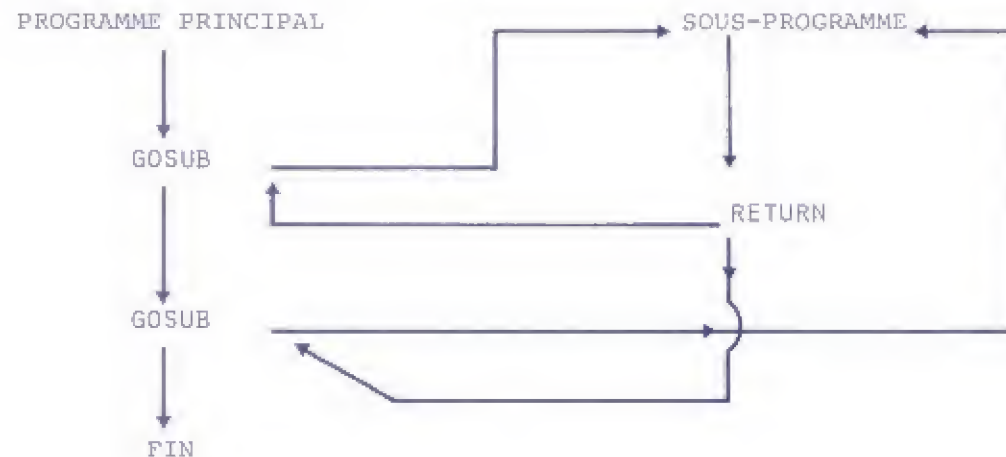
```


CHAPITRE 14

LES SOUS-PROGRAMMES (GOSUB)

Lors de l'élaboration d'un programme, vous pouvez avoir besoin de faire appel plusieurs fois à la même instruction (ou à la même série d'instructions). Vous pouvez (et c'est probablement ce que vous avez fait jusqu'à maintenant) écrire autant de fois qu'il vous est nécessaire cette ou ces instructions; mais, pour des raisons bien compréhensibles, nous allons écrire cette instruction une seule fois et l'appeler par l'instruction GOSUB.

Cette commande, suivie d'un n° de ligne et placée dans le programme principal, indiquera à l'ordinateur d'aller au n° indiqué, d'exécuter la ou les instructions contenues dans ce sous-programme, puis, après les avoir exécutées, de retourner à la ligne suivante du programme principal. Ceci lui étant ordonné par la commande RETURN (attention, ne pas confondre avec la touche RETURN) qui doit être tapée lettre par lettre sur le clavier et qui doit IMPÉRATIVEMENT terminer un sous-programme GOSUB.



Ex: étant donné 3 listes A, B, C de 20 éléments chacune, nous désirons:

- 1 - faire le produit élément par élément des registres $A * B$, $B * C$, et $C * A$ et mettre ce résultat dans le registre T.
- 2 - chercher à chaque passage le maximum des nombres du registre T et l'imprimer s'il est supérieur à 10.

Si nous ne désirons pas de sous-programmes, nous pouvons écrire:

```
NEW
AUTO 10, 5
10 DIM A (20), B (20), C (20), T (20)
15 FOR X = 1 TO 20
```

```

20 INPUT A (X), B (X), C (X)
25 T (X) = A (X) * B (X)
30 NEXT X
35 Y = T (1)
40 FOR X = 2 TO 20
45 IF T (X) <= Y THEN 55
50 Y = T (X)
55 NEXT X
60 IF Y <= 10 THEN 70
65 PRINT "MAX DE A PAR B =", Y
70 FOR X = 1 TO 20
75 T (X) = B (X) * C (X)
80 NEXT X
85 Y = T (1)
90 FOR X = 2 TO 20
95 IF T (X) <= Y THEN 105
100 Y = T (X)
105 NEXT X
110 IF Y <= 10 THEN 120
115 PRINT " MAX DE B PAR C =", Y
120 FOR X = 1 TO 20
125 T (X) = C (X) * A (X)
130 NEXT X
135 Y = T (1)
140 FOR X = 2 TO 20
145 IF T (X) <= Y THEN 155
150 Y = T (X)
155 NEXT X
160 IF Y <= 10 THEN 170
165 PRINT "MAX DE C PAR A =", Y
170 END

```

Les séquences des lignes 15 à 30, 70 à 80, et 120 à 130 calculent les produits des registres A, B et C.

Celles des lignes 35 à 65, 85 à 115 et 135 à 165 sont analogues, en ce sens qu'elles calculent le maximum stocké en registre T, lequel contient suivant le cas le produit des registres $A * B$, $B * C$ ou $C * A$ et impriment ce chiffre quand il est supérieur à 10.

- Or, ce traitement est identique dans les trois cas de figure, et c'est là que l'utilité d'un sous-programme unique, appelé par un GOSUB, se fait sentir!

Voici donc le même programme réécrit avec GOSUB:

```
5   DIM A (20), B (20), C (20), T (20)
10  FOR X = 1 TO 20
15  INPUT A (X), B (X), C (X)
20  T (X) = A (X) * B (X)
25  NEXT X
30  GOSUB 100
35  FOR X = 1 TO 20
40  T (X) = B (X) * C (X)
45  NEXT X
50  GOSUB 100
55  FOR X = 1 TO 20
60  T (X) = C (X) * A (X)
65  NEXT X
70  GOSUB 100
75  END

100 Y = T (1)
110 FOR X = 2 TO 20
120 IF T (X) <= Y THEN 140
130 Y = T (X)
140 NEXT X
150 IF Y <= 10 THEN 170
160 PRINT Y
170 RETURN
```

LES SOUS-PROGRAMMES EMBOÎTÉS

Nous venons d'examiner l'utilisation de GOSUB, mais il existe d'autres applications de cette instruction.

En effet, à l'intérieur d'un sous-programme, on peut appeler un autre sous-programme au moyen d'une instruction GOSUB ou enchaîner deux différentes parties d'un sous-programme en les appelant à l'aide d'un test ou d'un GOTO.

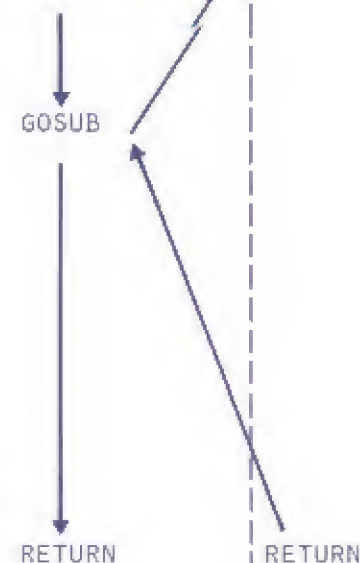
PROGRAMME
PRINCIPAL



1ER NIVEAU DE
SOUS-PROGRAMME



2EME NIVEAU DE
SOUS-PROGRAMME



Exemple:

```
10  INPUT A
20  GOSUB 3000
30  IF A = 10 THEN 50
```

} programme principal

```
100  END
3000 INPUT B
3010 IF B > 0 THEN 3060
3020 B = 2 * A/B
3030 PRINT "B =", B
3040 RETURN
3050 B = B + 1
3060 INPUT C
3070 B = A/B + C
3080 RETURN
```

} sous-programme

A la ligne 20 l'ordinateur est envoyé en 3000 pour exécution. Puis si la valeur de B est supérieure à 0, il va directement en ligne 3060 dans la seconde partie du sous-programme où il exécute les instructions des lignes 3060 et 3070 avant de retourner au programme principal (c'est-à-dire à la ligne 30). Si le test est négatif, il exécute les instructions des lignes

De même, l'appel d'un sous-programme peut renvoyer à une ligne différente de la première ligne du sous-programme, comme dans l'exemple ci-dessous:

```

5      NEW
10     INPUT A, B
20     GOSUB 3000
30     B = A + B
40     GOSUB 3010
"
"
"
"

100    END
3000   A = A + 1
3010   IF A * B > 0 THEN 3030
3020   A = 2 * A + B
3030   PRINT (A - B)/2
3040   RETURN

```

A la ligne 20, l'ordinateur exécute l'instruction de la ligne 3000, alors qu'il ne le fait pas au second appel (ligne 40).

INTERDICTIONS

Un sous-programme ne peut pas en appeler un autre après avoir lui-même été appelé par cet autre; et un sous-programme ne peut évidemment pas s'appeler lui-même.

L'exemple ci-dessous est incorrect:

```

10
-
-
100    END
      } programme principal

1000   INPUT A
1010   IF A >= 0 THEN 1030
1020   GOSUB 1050
1030   A = 2 * B/A
1040   RETURN

```

```

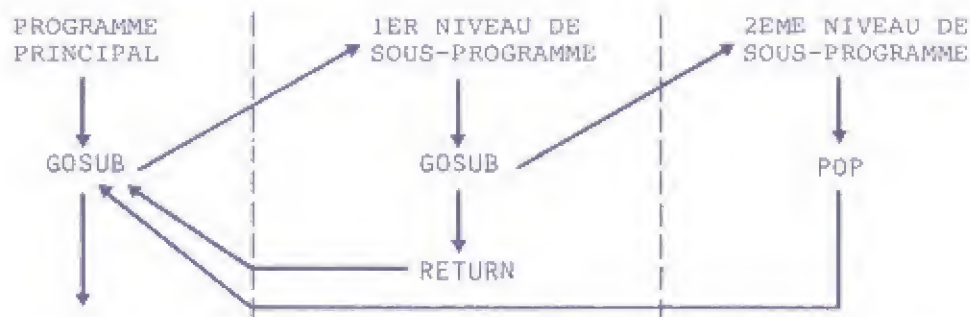
1050 INPUT C
1060 A = B/A + C
1070 GOSUB 1000
1080 RETURN

```

De plus, il est interdit de composer un programme comportant plus de 16 sous-programmes emboîtés.

LA COMMANDE POP

Cette commande permet de sauter un niveau de retour de sous-programme:



Entrez le simple programme ci-après qui vous en fera la démonstration:

```

NEW
10 A = 1 : B = 1 : C = 1 : D = 1
20 GOSUB 100
30 PRINT "A, B, C, D..." ; A ; B ; C ; D
40 END                                     } programme principal

100 A = A + 1
110 GOSUB 200
120 PRINT "A =" ; A
130 RETURN                               } 1er sous-programme

200 B = B + 1
210 GOSUB 300
220 B = B + 1
230 PRINT "B =" ; B
240 RETURN                               } 2ème sous-programme

```

```

300 C = C + 1
310 GOSUB 400
320 C = C + 2
330 PRINT "C =" ; C
340 RETURN

400 D = D + 4
410 PRINT "D =" ; D
420 RETURN

```

et...RUN

} 3ème sous-programme

} 4ème sous-programme

Vous verrez afficher ceci sur l'écran:

```

D = 5
C = 4
B = 3
A = 2
A, B, C, D,,, 2345

```

- (Si vous ne comprenez pas le mécanisme, rappelez-vous: TRACE affichera toutes les lignes du programme dans leur ordre d'exécution).

Maintenant, supposons que nous ne voulions pas changer la valeur de C, sans modifier le 3ème sous-programme! Nous ajoutons simplement:

```

415 POP

```

et... RUN

Le résultat sera celui-ci:

```

D = 5
B = 3
A = 2
A, B, C, D,,, 2 3 (2) 5

```

- L'effet de POP est donc bien celui voulu, qui est de "sauter" un niveau de retour (au lieu de revenir en 320, l'exécution revient en 230). Bien entendu, selon le résultat que vous désirez obtenir, vous pouvez écrire POP soit en ligne 415, soit en 335, soit en 235... mais attention: pas en 125, car vous ne pouvez l'utiliser au sein d'un seul sous-programme (il en faut au moins deux).

Toujours dans le programme ci-dessus, supposons que vous désiriez revenir directement au programme principal SANS exécuter les trois sous-programmes précédents. Vous aurez à modifier la ligne 420 comme suit:

420 GOTO 30

Le résultat sera:

D = 5

A, B, C, D... (2) (2) (2) 5

Par contre, vous ne pouvez vous contenter d'écrire 415 POP et... rien d'autre, car dans ce cas vous aurez:

D = 5

***NO END ERR

N'en déduisez pas pour autant que vous pouvez écrire 420 END, car vous auriez ceci:

D = 5 ... qui est un piètre résultat pour un tel programme!

EXERCICE XIV/1

GOSUB

Ecrivez un programme qui:

- rende blanc tout l'écran
- dessine 2 symboles, de couleurs différentes, et qui les fasse avancer sur quatre rangées différentes.

Mettez les programmes des dessins dans des sous-programmes différents.

```
10 GR
20 COLOR = 15
30 FOR Y = 0 TO 39
40 HLIN 0, 39 AT Y : NEXT Y
50 FOR Y = 3 TO 33 STEP 10
60 FOR X = 1 TO 35 STEP 2
70 COLOR = 2 : GOSUB 200
80 COLOR = 15 : GOSUB 200
90 COLOR = 4 : GOSUB 300
100 COLOR = 15 : GOSUB 300
110 NEXT X
120 NEXT Y
130 END
```

```
200 FOR N = 0 TO 4
210 HLIN X, X + 2 AT Y + N
220 NEXT N
230 FOR P = 0 TO 20 : NEXT P
240 RETURN
300 PLOT X + 2, Y
310 FOR N = 1 TO 3
320 HLIN X + 1, X + 3 AT Y + N
330 NEXT N
340 PLOT X + 2, Y + 4
350 FOR P = 0 TO 20 : NEXT P
360 RETURN
```

Effacez les lignes 80 et 100 pour voir exactement comment les dessins avancent!

CHAPITRE 15

LEVIER DE COMMANDE (GRAPHIQUES, JEUX)

(Ce "levier" muni de son câble de raccordement est fourni seulement en option!)

Il existe une fonction PDL qui contrôle ce levier (voir sa définition dans le glossaire) pour laquelle nous allons vous donner des exemples d'utilisation. Mais au préalable, il est nécessaire d'enficher le connecteur situé en bout du cordon qui part du boîtier du levier, dans le support prévu à cet effet à l'intérieur de votre ordinateur (en haut et à droite), et marqué GAME I/O (les encoches situées sur la partie arrière du coffret sont utilisées pour le passage du cordon, et permettent de refermer le couvercle).

Nous savons que le fait de demander l'affichage de PDL, suivi d'un Ø ou 1 entre parenthèses, donnera une valeur décimale comprise dans la gamme de Ø à 255.

Ex: PRINT PDL (Ø)

et suivant la position de votre levier, vous verrez apparaître un nombre.

Pour mieux comprendre ce phénomène, entrez ceci:

```
1Ø   A = PDL (Ø)
2Ø   PRINT A
3Ø   GOTO 1Ø
```

et manœuvrez le levier!... Vous voyez apparaître une succession ininterrompue de nombres (toujours entre Ø et 255). Nous allons maintenant les faire correspondre à une position relative sur l'écran.

Pour cela, écrivons un programme qui nous permettra de "promener" une tache lumineuse sur l'écran:

```
1Ø   GR : COLOR = 4   (initialise mode graphique: sélectionne une couleur)
2Ø   X = PDL (Ø)       (affecte à X la valeur PDL (Ø))
3Ø   IF X > 239 THEN X = 239
4Ø   X = X/6
5Ø   Y = PDL (1)
6Ø   IF Y > 239 THEN Y = 239
7Ø   Y = Y/6
8Ø   PLOT X, Y
9Ø   GOTO 1Ø
```

Les lignes 3Ø et 4Ø permettent de ramener la gamme des nombres obtenus par le levier (de Ø à 255) aux limites autorisées par le mode graphique. La ligne 3Ø signale à X que chaque fois qu'il atteindra la valeur 239, il sera égal à celle-ci. La ligne 4Ø affecte une nouvelle valeur à X en divisant 239 par 6, soit 39, qui correspond à la gamme autorisée (Ø à 39).

La même instruction est donnée à Y aux lignes 60 et 70 (bien sûr, X = abscisses, Y = ordonnées), puis la ligne 80 demande l'affichage successif des "taches" sur lesquelles on passe dans la couleur choisie en ligne 10. Enfin la ligne 90 recommence indéfiniment le programme.

- (Attention: ne jamais porter sur la même ligne d'instruction les initialisations des valeurs de PDL 0 et 1).

Jouez un moment avec votre levier... Intéressant n'est-ce pas?

Vous pourriez obtenir un effet plus heureux avec une seule modification! Changez la ligne 10 comme suit: GR : COLOR = RND (15) + 1

Votre ordinateur sélectionnera une nouvelle couleur d'une manière totalement aléatoire, chaque fois qu'il recommencera le programme.

Allons plus avant! Jusqu'à présent vous n'obteniez que des "taches" successives. Comment tracer maintenant des lignes continues? Rien de plus facile. Ajoutez au programme:

5 GR

puis modifiez la ligne 10: 10 COLOR = RND (15) + 1

Joli n'est-ce pas!

C'est le même principe qui est utilisé dans les jeux (tel que: MUR DE BRIQUES).

N.B.: sur votre boîtier figurent deux "boutons". Pour le moment rien ne se passe si vous y touchez! L'explication nécessaire pour les utiliser est donnée au chapitre n° XVI (POKE, PEEK).

EXERCICE XV/1

PDL (0)

PDL (1)

SCRN (X, Y)

Ecrivez un programme pour un dessin coloré quelconque.

Mettez les renseignements nécessaires pour reproduire ce dessin en mémoire (registre).

Effacez l'écran graphique et faites reparaitre le dessin à l'aide du levier de commande. (Le contenu de PDL (0) et celui de PDL (1) vous indique un point. Retrouvez la couleur de ce point dans le registre et affichez l'ensemble).

```
10   DIM A (1600)
20   GR : X1 = 2 : X2 = 38 : Y1 = 2 : Y2 = 38
30   FOR C = 1 TO 10
40     COLOR = C
50     FOR Y = Y1 TO Y2
60       HLIN X1, X2 AT Y
70     NEXT Y
80     X1 = X1 + 2 : X2 = X2 - 2 : Y1 = Y1 + 2 : Y2 = Y2 - 2
90   NEXT C
```

```
200  N = 1 : PRINT "JE TRAVAILLE !!!"
210  FOR Y = 0 TO 39
220    FOR X = 0 TO 39
230      A (N) = SCRN (X, Y)
240      N = N + 1
250    NEXT X : NEXT Y
```

```
300  GR : CALL -936
310  Y = PDL (1)
320  IF Y > 239 THEN Y = 239 : Y = Y / 6
330  X = PDL (0)
340  IF X > 239 THEN X = 239 : X = X / 6
350  N = Y * 40 + X + 1
360  COLOR = A (N)
370  PLOT X, Y
380  GOTO 310
```


Avant de vous donner des explications détaillées sur les instructions CALL, POKE, PEEK, il est nécessaire que nous vous disions quelques mots concernant la manière dont fonctionne un ordinateur.

Jusqu'à maintenant nous n'avons traité que le langage conversationnel appelé BASIC, mais ceci n'est que le "sommet de l'iceberg".

En effet votre ordinateur possède un autre langage qui est le langage machine. Ce langage est en binaire (suite de 0 et de 1) et il est le seul langage que comprenne votre ordinateur.

Lorsque vous exécutez un programme en BASIC, l'ordinateur traduit le programme en langage machine. Il transmet ces instructions-machine au microprocesseur (ROCKWELL 6502), qui exécute le programme et vous permet d'obtenir vos résultats.

Votre ordinateur a la possibilité de mélanger le BASIC et le langage machine, c'est-à-dire que dans un programme BASIC vous pouvez modifier la mémoire (PEEK, POKE), ou faire exécuter un sous-programme en langage machine (CALL).

Nous ne vous en dirons pas plus pour le moment, car tout ce qui concerne le langage machine sera expliqué dans un second manuel, consacré au BASIC évolué. (PALSOF)

CALL:

C'est un sous-programme (ce en quoi on l'assimile à un GOSUB), mais la différence réside dans le fait qu'il appelle un sous-programme en langage machine alors que le CALL lui-même s'écrit à l'intérieur du programme en BASIC. Au lieu d'être suivi d'un numéro de ligne comme GOSUB, CALL est suivi d'un nombre décimal qui représente un endroit (adresse) dans la mémoire.

L'instruction que vous rencontrez couramment en BASIC est la suivante:

CALL -936

qui peut être utilisée aussi bien en mode d'affichage immédiat que différé.

Quand l'ordinateur rencontre cette instruction, il va examiner à l'emplacement mémoire -936 ce qui s'y trouve et exécute l'instruction qui dans ce cas lui demande d'effacer de l'écran toute inscription avant de passer aux lignes suivantes.

Vous obtenez le même résultat à l'aide de la touche  , vous vous rappelez?

Mais on ne peut l'utiliser à l'intérieur d'un programme.

Il existe bien d'autres instructions CALL (voir chapitre XVII). Elles seront détaillées dans un second manuel.

- Puisque nous venons d'évoquer le langage machine à propos de CALL, nous

pouvons également vous donner quelques explications concernant POKE et PEEK. Vous en trouverez la désignation dans votre glossaire, mais nous pouvons l'exprimer par les phrases suivantes:

- Stocker une information à un emplacement mémoire défini se fait à l'aide de la commande POKE.
- Appeler une information à un emplacement mémoire défini se fait à l'aide de la commande PEEK.

● Et à ce propos, revenons au mode Graphique! (tapez GR et lisez la suite...)

Nous avons vu, au chapitre IV, que la commande GR initialisait le mode graphique mixte, c'est-à-dire de 0 à 39, laissant quatre lignes de texte disponibles dans le bas de l'écran. Jusqu'ici, nous n'avons rien dit du mode graphique intégral (qui alloue l'intégralité de l'écran au graphique, sans texte apparent).

Comment l'obtenir? Eh bien, en utilisant la commande POKE! L'ordinateur, quand il rencontrera l'instruction POKE -16302,0, saura qu'il faut initialiser le mode graphique intégral, de même que l'instruction POKE -16301,0, redonnera les quatre lignes de texte (dans ce cas, vous pouvez indifféremment taper POKE -16301,0, ou GR, le même effet sera obtenu). Essayez. De même, nous avons vu que pour repasser en mode TEXTE, nous pouvons utiliser la commande TEXT. Son équivalent est POKE -16303,0 suivi de POKE 34,0 qui donne le même résultat.

● Maintenant, vous désirez peut-être attirer l'attention sur un point particulier d'un texte. Pour ce faire, entrez POKE 50,63 juste avant l'instruction concernant ce point particulier dans votre programme. Cela provoquera l'inversion de la vidéo et affichera la partie prévue en noir sur fond blanc (attention: l'inversion ne concerne que les affichages venant de l'ordinateur et non ceux que vous tapez). Si vous avez plusieurs parties de votre texte à détacher ainsi et que parmi celles-ci l'une d'entre elles soit plus importante, alors tapez POKE 50,127. Cela provoquera une alternance de fond noir sur fond blanc (attention: le clignotement n'est effectif uniquement que sur les lettres et le signe ^).

Bien entendu, dès que vous voulez revenir à l'affichage normal, vous entrez: POKE 50,255.

● Voulez-vous inscrire (du texte ou un graphique) dans une partie plus restreinte de l'écran?
C'est tout-à-fait possible!

En effet, normalement la fenêtre de visualisation est initialisée ainsi:

* côté gauche (de l'écran)	= 0	Instruction: POKE 32, expr.
côté droit	" = 40	Instruction: POKE 33, expr.
sommet	" = 0	Instruction: POKE 34, expr.
base	" = 24	Instruction: POKE 35, expr.

C'est-à-dire que tout le champ de votre écran TV est utilisé par l'ordinateur.

Supposons que vous vouliez réduire la fenêtre de votre écran TV en le limitant entre les colonnes 10 et 20 et les lignes 10 et 20.

Vous écrirez dans votre programme:

```
POKE 32, 10
POKE 33, 20
POKE 34, 10
POKE 35, 20
```

Puis, pour inclure le curseur à l'intérieur de ce rectangle, ajoutez
CALL -936 et... c'est tout!

Pour revenir à l'intégralité de l'écran, changez les valeurs ci-dessus
dans votre programme, en les remplaçant par les valeurs de*.


LES SONS

Vous savez que le fait d'appuyer sur  fait résonner un BIP! Vous



pouvez obtenir le même résultat soit avec l'instruction PEEK (-16336) ,
soit POKE -16336,0, mais ce son bref n'est guère attrayant et nous verrons
un peu plus loin comment faire... de la musique!

CONTRÔLE CLAVIER





Jusqu'ici, le seul moyen pour arrêter un programme en cours de déroulement
était un , et pour le reprendre taper CONTINUE. Mais à ce stade de



vosre initiation, vous désirez sûrement pouvoir stopper un programme sans
être obligé d'en sortir! Il existe une instruction double pour cela...

```
IF PEEK (-16384) < 128 THEN expr.
    POKE -16368,0
```

En voici l'explication:

À l'emplacement mémoire de -16384, la valeur qui s'y trouve est normale-
ment inférieure à 128. Quand n'importe quelle touche du clavier est enfon-
cée (à l'exception de , ,  ou ) , cette valeur
change et devient supérieure à 127. Puis l'instruction POKE -16368,0 "lit"
le clavier et lui rend son autonomie.

Cet ensemble, relié au programme par un test, permet d'interrompre le
déroulement de celui-ci. Puis, en appuyant sur une touche quelconque du
clavier (la barre d'espacement est bien pratique pour cela!), le program-
me recommencera à son début.

● Voici un exemple très simple:

```
10 GR
20 FOR X = 0 TO 39 : COLOR = RND (15) + 1
30 FOR Y = 0 TO X
40 PLOT X, Y
```



```

50  NEXT Y
60  NEXT X
70  GOTO 20
    RUN...

```

Si vous voulez arrêter un tel programme, il faut utiliser un **CTRL** puis

C

RUN (ou CON) pour recommencer.

Modifions le programme comme suit:

lignes 10 à 60... inchangées

```

70  IF PEEK (-16384) < 128 THEN 70
80  POKE -16368,0
90  GOTO 20

```

La ligne 70 indique à l'ordinateur d'arrêter le programme dans l'attente d'une activation du clavier.

Dans ce cas la ligne 80 rend le contrôle au clavier et passe en 90 pour recommencer le programme.

- Ce programme s'arrête donc automatiquement lorsque la ligne 70 est rencontrée, mais vous pouvez déclencher cet arrêt quand vous le désirez. Il suffit alors d'écrire:

lignes 10 à 60... inchangées

```

70  IF PEEK (-16384) < 128 THEN 20
80  POKE -16368,0
90  IF PEEK (-16384) < 128 THEN 90
100 POKE -16368,0
110 GOTO 20

```

Ainsi, vous appuierez sur une touche pour arrêter le programme, et appuierez de nouveau pour continuer, et cela indéfiniment...

- Au chapitre relatif au "LEVIER DE COMMANDE", nous avons évoqué les touches qui se trouvent sur votre boîtier. Pour les activer, il existe des instructions dont le fonctionnement est le suivant:

PEEK (-16287) pour PDL (0) (qui correspond à X = abscisses)

et PEEK (-16286) pour PDL (1) (qui correspond à Y = ordonnées)

On peut, dans un programme, utiliser une variable d'initialisation comme ceci:

```

X0 = PEEK (-16287)
IF X0 <= 127 THEN expr.
Y1 = PEEK (-16286)
IF Y1 <= 127 THEN expr.

```


Evidemment, vous pouvez utiliser n'importe quelle variable numérique à la place de $x0$ et $y1$.

Voici un programme qui démontre une utilisation des touches $x0$ et $y1$:

```
10 CALL -936
20  $x0 = \text{PEEK} (-16287)$ 
30  $y1 = \text{PEEK} (-16286)$ 
40 VTAB 10
50 IF  $x0 \leq 127$  THEN 70
60 TAB 10 : PRINT "ZERO"
70 IF  $y1 \leq 127$  THEN 10
80 TAB 25 : PRINT  $x0$  UN"
90 GOTO 10
```

et amusez-vous avec les touches...!

LES TONS, COMMENT LES GÉNÉRER :

Votre micro-ordinateur ITT possède un amplificateur muni d'un haut-parleur.

Nous avons vu précédemment que l'instruction `PEEK -16336` produisait un son bref (BIP).

Elle n'est donc pas suffisante pour produire des tonalités, car pour cela il est nécessaire d'établir une fréquence et une durée, c'est-à-dire: activer le haut-parleur à des intervalles donnés et pour des durées fixées.

Il existe des emplacements mémoire où peuvent être stockées la fréquence relative (F) entre les émissions du haut-parleur, ainsi que la valeur de la durée relative (D).

F comme D se situent dans une gamme d'utilisation de 0 à 255, et une fois définies les valeurs de F et de D, elles seront appelées par l'instruction `CALL 2`. Cela déclenchera l'émission sonore du haut-parleur, puis un laps de temps égal à la valeur définie en D, et ainsi de suite jusqu'à ce que toutes les valeurs aient été lues.

- Il est important de noter que c'est la valeur F qui influe sur la valeur D. En d'autres termes, plus la valeur de F est grande (vers 255), plus la durée d'écoute sera longue, quelle que soit la valeur de D.

Les possibilités sont relativement importantes, puisque vous pouvez générer $255 \wedge 2$ sons, ce qui donne finalement 255 tonalités différentes. Bien sûr, ces chiffres sont théoriques, car les fréquences hautes et les fréquences basses sont inaudibles à l'oreille humaine.

Voici un petit programme qui illustre ces explications:

```
5 POKE 2,173 : POKE 3,48 : POKE 4,192 :
  POKE 5,136 : POKE 6,208
```

} Ces instructions doivent
être entrées (comme initia-

```

10 POKE 7,4 : POKE 8,198 : POKE 9,1 :      lisation) à chaque recher-
   POKE 10,240 : POKE 11,8                  che d'un programme de...
15 POKE 12,202 : POKE 13,208 : POKE         } musique.
   14,246 : POKE 15,166 : POKE 16,0
20 POKE 17,76 : POKE 18,2 : POKE 19,0 :
   POKE 20,96
30 F = 35 : D = 60
40 POKE 0,F : POKE 1,D : CALL 2
50 GOTO 30
60 END

```

Mais ce programme ne génère qu'une tonalité en rapport avec la fréquence et la durée définies en lignes 30. Il faut donc, pour obtenir une "musique" le relier à une ou plusieurs boucles FOR ... NEXT, en variant à chaque fois les valeurs de F et D selon le ton recherché.

Nous ne vous en dirons pas plus car, ceci étant affaire de goût et d'imagination créatrice, nous vous laissons le soin d'effectuer des essais suivant les formules données ci-dessus. Nous vous indiquons seulement, ci-après, un programme qui génère les premières mesures d'une symphonie célèbre. Essayez le...

```

5 POKE 2,1/3 : POKE 3,48 : POKE 4,192 : POKE 5,136 : POKE 6,208
10 POKE 7,4 : POKE 8,198 : POKE 9,1 : POKE 10,240 : POKE 11,8
15 POKE 12,202 : POKE 13,208 : POKE 14,246 : POKE 15,166 : POKE 16,0
20 POKE 17,76 : POKE 18,2 : POKE 19,0 : POKE 20,96
60 FOR A = 1 TO 3 : F = 35 : D = 60 : GOSUB 200 : NEXT A
70 F = 44 : D = 200 : GOSUB 200
80 FOR A = 1 TO 3 : F = 40 : D = 60 : GOSUB 200 : NEXT A
90 F = 47 : D = 200 : GOSUB 200
100 FOR A = 1 TO 3 : F = 35 : D = 60 : GOSUB 200 : NEXT A
110 F = 45 : D = 60 : GOSUB 200
120 FOR A = 1 TO 3 : F = 34 : D = 60 : GOSUB 200 : NEXT A
130 F = 35 : D = 60 : GOSUB 200
140 FOR A = 1 TO 3 : F = 22 : D = 60 : GOSUB 200 : NEXT A
150 F = 26 : D = 200 : GOSUB 200
160 END
200 POKE 0,F : POKE 1,D : CALL 2 : RETURN

```

A vous de jouer!...

EXERCICE XVI/1

PEEK, POKE

Changez la première partie du programme de l'exercice XV/1 (lignes 20 à 90) pour pouvoir utiliser le levier.

Donnez la possibilité de changer la couleur (de 0 à 9) et d'aller dans la deuxième partie du programme (mise en mémoire du dessin).

```
10 DIM A$(5), A(400)
20 GR
30 INPUT COULEUR = , C : COLOR = C
40 X = PDL (0) : IF X > 239 THEN X = 239 : X = X / 6
50 Y = PDL (1) : IF Y > 239 THEN Y = 239 : Y = Y / 6
60 PLOT X, Y
70 IF PEEK (-16384) < 128 THEN 40 : POKE -16368, 0
80 INPUT VOULEZ-VOUS CHANGER LA COULEUR? (O/N) , A$
90 IF A$(1,1) = O THEN 30
```

Les lignes 200 à 380 restent inchangées!

QUELQUES MOTS SUR LA CONSTRUCTION D'UN PROGRAMME :

Quand vous parviendrez à une maîtrise suffisante de votre ordinateur et que vos programmes seront de plus en plus élaborés, vous éprouverez sans doute le besoin de chercher des moyens de le faire fonctionner plus rapidement. Voici quelques uns de ces moyens:

- A- Supprimer les REMarques ou les placer en fin de programme après la ligne END.
- B- Placer les sous-programmes (GOSUB) avant le programme principal, en prenant soin de les placer par ordre décroissant d'utilisation.
- C- Utiliser autant que possible une seule lettre pour les variables. Egalement, lors de leur initialisation, les placer dans l'ordre décroissant d'utilisation. Réduire au maximum le nombre des variables. Réutiliser les mêmes variables chaque fois que possible.
- D- Utiliser des boucles fermées (FOR ... NEXT) au lieu de boucles ouvertes.
- E- Mettre le maximum possible d'instructions sur une même ligne en les séparant par des :
- F- Calculer les sous-expressions en une seule fois au lieu de le faire à chaque fois qu'elles sont demandées. Il est préférable d'écrire:
$$D = A(B) : \text{IF } D \neq 0 \text{ THEN } C = D * D/5$$
plutôt que:
$$\text{IF } A(B) \neq 0 \text{ THEN } C = (A(B) * A(B))/5$$
Le gain de temps et de place en mémoire est appréciable, surtout dans un long programme.
- G- Supprimer les parenthèses autant que possible, en suivant les règles de priorité des opérateurs. (Bien sûr, cela ira plus vite mais aux dépens de la compréhension).
- H- Si un sous-programme (GOSUB/RETURN) est affiché seulement une fois dans un programme, écrivez-le comme une partie intégrante du programme principal, car les GOSUB prennent beaucoup de temps. De même, si le sous-programme est court et est affiché deux ou trois fois seulement, il vaut mieux, là aussi, l'écrire (deux ou trois fois) dans le programme principal.
- I- D'une manière générale, les parties d'un programme qui sont exécutées seulement une fois, n'ont pas besoin d'être "comprimées". Réservez votre attention pour celles qui sont répétitives, dans une boucle. En résumé, il faut considérer que raccourcir un programme, soit pour conserver de la mémoire, soit pour une exécution plus rapide, aura pour conséquence une compréhension plus difficile de la lecture de ce programme.

N.B.: en ce qui concerne les programmes comportant de nombreux GOSUB, le fait de les supprimer pour accroître la rapidité d'exécution s'effectue au détriment de la mémoire disponible.

CHAPITRE 17

GRAPHIQUES EN COULEUR A HAUTE RESOLUTION

Le micro-ordinateur ITT 2020 dispose d'un mode d'affichage graphique couleur à haute résolution de 360 points x 192 points. (Appelé ici HIRES).

Un minimum de 16 K RAM est nécessaire. La programmation en HIRES peut s'effectuer de trois manières différentes:

- 1° - En utilisant le langage BASIC évolué à virgule flottante, les commandes HIRES étant intégrées à ce langage. (Se référer au manuel BASIC évolué à virgule flottante).
- 2° - En utilisant le langage BASIC intégré à l'ITT 2020.
- 3° - En utilisant le langage machine de microprocesseur 6502.

Ce chapitre traite des langages (2) et (3).

1 - UTILISATION DES SOUS-PROGRAMMES HIRES :

La première instruction d'un programme BASIC utilisant les sous-programmes HIRES doit être:

`0 X0 = Y0 = COLR = SHAPE = ROT = SCALE`

Le but de cette instruction est de déclarer le nom des six premières variables BASIC dans un ordre constant. Lors de l'exécution du programme, chacun de ces six paramètres se verra stocké à des locations mémoire fixes par rapport à l'adresse contenue dans le pointeur de début des variables BASIC LOMEM, rendant ces paramètres accessibles aux sous-programmes machine HIRES.

Après que le nom des paramètres ait été déclaré, les noms des sous-programmes HIRES peuvent eux-même être déclarés en assignant au nom du sous-programme un nombre représentant son adresse de départ.

Appeler les sous-programmes par leur nom est préférable à les appeler par leur adresse de départ, celle-ci étant susceptible d'être modifiée dans des versions futures HIRES et leur nom étant plus compréhensible à la lecture des programmes.

- L'instruction suivante donne les adresses de départ de tous les sous-programmes HIRES accessibles depuis le BASIC. Il est évident que l'utilisateur peut appeler ces sous-programmes par d'autres noms:

```
5 INIT   = 2048 : CLEAR = 2062 : BKGND = 2834 :  
  POSN   = 2763 : PLOT  = 2790 : LINE   = 2796 :  
  DRAW   = 2850 : DRAW1 = 2853 : DRAW   = 2840 :  
  XDRAW1 = 2843 : FIND  = 2605
```

Les définitions de variables inutiles devront être évitées car celles-ci ralentissent le déroulement du programme.

Les valeurs permises assignables aux couleurs sont les suivantes:

Noir = 0 ou 128 bleu = 42

Blanc = 127 ou 255 vert = 85

Les valeurs 170 et 213 peuvent aussi générer les couleurs cyan et rouge.

Dans les cas extrêmes, les techniques suivantes amélioreront la vitesse de déroulement du programme.

- 1° - Omettre de déclarer le nom des sous-programmes et les appeler directement par leur adresse de départ -et non par leur nom-. Ceci ne s'applique pas aux paramètres.
- 2° - Déclarer le nom des variables le plus fréquemment utilisées avant de déclarer le nom des sous-programmes. L'exemple ci-dessous accélérera le déroulement d'un programme utilisant les variables I, J et K d'une manière répétitive:
0 X0 = Y0 = COLR = SHAPE = ROT = SCALE
2 I = J = K
5 INIT = 2048 : CLEAR = 2062 : ... ETC ...
- 3° - Utiliser le nom des paramètres comme nom de variable à chaque fois que cela est possible. Comme ils sont déclarés les premiers; ils sont les plus accessibles du BASIC.

2 - SOUS-PROGRAMMES D'INITIALISATION :

L'affichage HIRES comprend normalement 360 points (en horizontal) sur 160 (en vertical) avec, en bas de l'écran, 4 lignes réservées au texte, il est initialisé par la commande BASIC:

CALL INIT

Le sous-programme INIT efface aussi l'affichage HIRES et initialise les autres sous-programmes HIRES. Après avoir appelé le sous-programme INIT, l'utilisateur peut substituer aux quatre lignes de texte, un affichage HIRES de 360 points x 192 points à l'aide de la commande suivante:

POKE -16302,0

Les 4 lignes de texte peuvent être récupérées à tout moment à l'aide de la commande:

POKE -16301,0

Les abscisses permises sont comprises entre 0 (complètement à gauche) et 359 (complètement à droite). Les ordonnées permises sont comprises entre 0 (en haut) et 159 ou 191 (en bas) suivant que l'on affiche ou non les 4 lignes de texte.

Après que INIT ait été appelé, on peut à tout moment effacer l'affichage HIRES par la commande:

CALL CLEAR

L'écran HIRES peut être coloré à l'aide du sous-programme BKGND.

Ce sous-programme nécessite que l'on ait précédemment assigné une couleur à la variable COLR.

L'exemple suivant génère un affichage HIRES entièrement bleu

```
0 X0 = Y0 = COLR
5 INIT = 2048 :BKGND = 2834
10 CALL INIT
20 COLR = 42
30 CALLBKGND
40 END
```

Si la variable COLR est supérieure à 255, le sous-programme BKGND produira un "BIP" dans le haut-parleur, le message ***RANGE ERR sera affiché et le programme s'arrêtera.

3 - POINTS ET LIGNES :

Le sous-programme PLOT s'utilise pour afficher un point isolé dans une couleur spécifiée précédemment. COLR doit être inférieure à 255, X0 compris entre 0 et 359, Y0 compris entre 0 et 191 quand PLOT est appelé, sinon un message d'erreur est affiché et le programme s'arrête.

Le programme suivant écrit un point blanc qui a pour abscisse 35 et pour ordonnée 55, puis un autre qui a pour coordonnées (85, 90)

```
0 X0 = Y0 = COLR
5 INIT = 2048 : PLOT = 2790
10 CALLINIT
20 COLR = 127
30 X0 = 35 : Y0 = 55 : CALL PLOT
40 X0 = 85 : Y0 = 90 : CALL PLOT
50 END
```

POSITIONNEMENT

Tracer une droite entre deux points est presque aussi simple qu'afficher un point. Après avoir affiché un dernier point par PLOT, comme dans l'exemple ci-dessus, un point suivant est défini par X0 = et Y0 = et le sous-programme LINE est appelé; COLR, X0 et Y0 doivent être dans les limites permises ci-dessus.

Le positionnement d'un point sur la page écran se fait par l'appel du sous-programme POSN.

Les coordonnées X0 et Y0 du point sont à spécifier (COLR n'est pas nécessaire). La location mémoire du point est calculée par POSN mais ce point n'est pas affiché sur la mémoire écran.

- Ce sous-programme se substitue à PLOT lorsque par exemple on veut dessiner une figure dont le point de départ n'est pas à afficher. (Déplacement seul). Voir le paragraphe figures.

Des lignes différentes qui se touchent peuvent être dessinées sans appeler à chaque fois le sous-programme PLOT (le point final d'une ligne devient le point de départ de la ligne suivante). L'exemple ci-dessous trace une ligne blanche du point ayant pour coordonnées (10, 20) au point (250, 70) puis une ligne bleue de ce point (250, 70) au point (20, 150) et une ligne verte de (20, 150) à (260, 30).

```
0 X0 = Y0 = COLR
5 INIT = 2048 : PLOT = 2790 : LINE = 2796
10 CALL INIT
20 COLR = 255 : X0 = 10 : Y0 = 20 : CALL PLOT
30 X0 = 250 : Y0 = 70 : CALL LINE
40 X0 = 20 : Y0 = 150 : COLR = 42 : CALL LINE
50 X0 = 260 : Y0 = 30 : COLR = 85 : CALL LINE
```

ATTENTION :

- Ne pas tenter de dessiner une ligne avant d'avoir fait appel à PLOT. En effet le point de départ de cette ligne n'étant pas spécifié, la ligne peut être dessinée dans n'importe quelle position de la mémoire, pas forcément dans la page HIRE (risques de détruire un programme, par exemple).

4 - FIGURES

A l'aide du langage BASIC intégré, jusqu'à 255 figures différentes peuvent être définies et stockées sur une seule bande magnétique. Se référer au manuel "Basic évolué à virgule flottante". Après avoir chargé les sous-programmes HIRE, une telle "bande de figures" (contenant un "fichier figures") peut être chargé comme suit:

- 1° - Introduire la bande de figure dans le magnétophone
- 2° - Charger les figures en RAM avec la commande BASIC
CALL 3000 (ou CALL SHLOAD)
- 3° - Mettre le magnétophone en position "REPRODUCTION"
- 4° - Attendre deux "BIP" successifs.

- Les fichiers figures se chargent toujours à partir de l'adresse hexadécimale 8C00 et les sous-programmes HIRE sont contenus entre 8800 et 8BFF. Après chargement du fichier figures le pointeur de début des variables BASIC (LOMEM) contient l'adresse suivant le dernier octet du fichier figures.

S'il n'y a pas assez de mémoire disponible pour charger le fichier figures, un "BIP" se fera entendre et le message:

***MEM FULL ERR
sera affiché.

Si aucun "BIP" ne se fait entendre lors du chargement du fichier figure,

il y a probablement un problème de connexion avec le magnétophone et il faut taper la touche `RESET` puis `Cc` (contrôl C) pour revenir au BASIC. Si l'on n'entend qu'un seul "BIP" et qu'il ne se passe plus rien, taper `RESET` et `Cc` et réinitialiser LOMEM à la valeur `3072` (3072) comme suit

LOMEM : 3072

Recommencer les opérations (1) à (4).

- 1 Le sous-programme DRAW permet de dessiner une quelconque des figures pré-définies contenues dans le fichier figure. Le point de départ, ou origine, de la figure est défini par `X0` et `Y0` et la couleur de la figure par `COLR`. Le numéro de la figure est défini par `SHAPE`. Par exemple: `SHAPE = 3` spécifie que l'on désire dessiner la 3^e figure du fichier. Un facteur de grossissement est défini par `SCALE` et une inclinaison par `ROT`.

`COLR` doit être compris entre 0 et 255, `X0` entre 0 et 359, `Y0` entre 0 et 191, `ROT` entre 0 et 255 (la valeur retenue par le sous-programme DRAW est modulo 64), `SCALE` entre 0 et 255 (0 est compris comme 256) et `SHAPE` entre 1 et le numéro de la dernière figure du fichier, sinon un message d'erreur est affiché et le programme s'arrête.

- 2 Autrement dit, le programmeur sera toujours informé lorsqu'il appellera un sous-programme HIRES faisant intervenir des paramètres non autorisés.

Le sous-programme XDRAW ne diffère de DRAW que par le fait que la figure sera traitée en "ou exclusif -XOR" avec le fond sur lequel elle est dessinée. L'opération XOR complémente tous les bits de la mémoire écran de la figure, les "0" devenant des "1" et vice versa. Aussi il n'est pas nécessaire de spécifier une couleur. Le sous-programme XDRAW a une particularité unique: deux appels successifs de ce programme dessinent la figure, puis l'efface.

Le programme suivant permet de faire pivoter la figure N° 3 du fichier, l'angle étant lié au palonnier 0. XDRAW est utilisé pour à la fois dessiner et effacer la figure. Bien que la couleur soit facultative, la variable `COLR` doit être déclarée, sinon les variables `SHAPE`, `ROT` et `SCALE` ne seront pas stockées à leur emplacement standard par rapport à LOMEM.

```
0   X0 = Y0 = COLR = SHAPE = ROT = SCALE
5   INIT = 2048 : XDRAW = 2840
10  CALL INIT
20  X0 = 140 : Y0 = 80 : SHAPE = 3 : SCALE = 2
30  R = 0 : GOTO 60 : REM DESSIN PREMIERE FIGURE
40  R = PDL (0) : IF R = ROT THEN 30
50  CALL XDRAW : REM EFFACER L'ANCIENNE FIGURE
60  ROT = R : CALL XDRAW : REM DESSINER NOUVELLE FIGURE
70  GOTO 40 : REM NOUVELLE INCLINAISON?
80  END
```

DRAW 1 et XDRAW 1 ne diffèrent respectivement de DRAW et XDRAW que par le fait que le dernier point écrit dans la mémoire écran sert de point de départ à la nouvelle figure. L'ancienne couleur reste en mémoire. Aussi X0, Y0 et COLR n'ont pas à être spécifiées à nouveau.

5 - SOUS-PROGRAMMES "FIND"

Il est possible de dessiner une figure et ensuite dessiner une ligne du dernier point de la figure jusqu'à un point déterminé. Après avoir dessiné la figure, il faut appeler le sous-programme FIND qui détermine les coordonnées X0 et Y0 du dernier point écrit. On écrit alors les coordonnées du point final de la ligne (X0 = ; Y0 =) et on appelle le sous-programme LINE.

6 - COMPTEUR DE COLLISIONS :

Des figures qui débordent les unes sur les autres définissent des "collisions", points que les figures ont en commun.

Le compteur de collision est situé à la location mémoire 810 (8 32A).

7 - CRÉATION D'UNE FIGURE :

La figure est codée sous forme de vecteurs qui sont stockés dans une suite d'octets en mémoire vive. Chaque octet de la figure est divisé en 3 vecteurs. Chaque vecteur spécifie si un point est ou non affiché sur l'écran et dans quel sens (haut, bas, droite, gauche) l'on doit se déplacer pour accéder au point suivant de la figure. Voici comment les 3 vecteurs sont organisés dans un octet :

Vecteur :	C	B	A
N° Bit :	7 : 6	5 : 4 : 3	2 : 1 : 0
Codage :	D D	P D D	P D D

Chaque paire de bits DD représente la direction d'un emplacement et chaque bit P représente le fait que le point soit affiché ou non, comme suit :

DD : 00 haut	↑	P = 0 point non affiché
01 droite	→	P = 1 point affiché
10 bas	↓	
11 gauche	←	

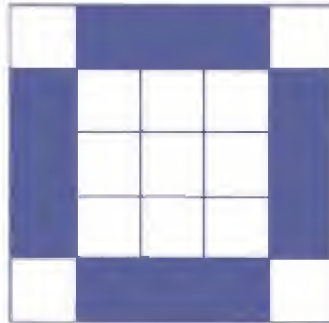
On notera que le vecteur C (bits les plus significatifs - MSB) ne possède pas de bit P. Par conséquent ce vecteur ne peut que représenter un déplacement sans affichage du point (P = 0 par défaut).

Les sous-programmes DRAW, DRAW 1, XDRAW, XDRAW 1 traitent les octets un par un et les vecteurs de chaque section dans l'ordre vecteur A, vecteur B, vecteur C. Quel que soit le vecteur d'un octet, si les vecteurs suivants (à gauche de ce dernier) ne contiennent que des "0", alors ces vecteurs

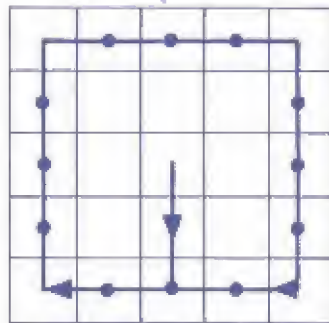
sont ignorés, c'est pourquoi l'octet ne peut pas se terminer (vecteur C) par un déplacement vers le haut (00): cet octet ne contenant que des zéros, il serait ignoré. De même si le vecteur C est 00, le vecteur B sera ignoré si il est lui-même égal à 000. Enfin si les 3 vecteurs sont nuls, les sous programmes DRAW interprètent cet octet comme la fin de la figure.

Un exemple simple fera sans doute la lumière sur ce qui vient d'être dit précédemment:

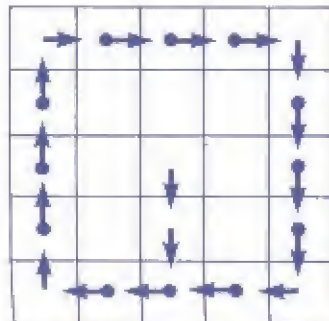
supposons que l'on désire dessiner la figure suivante:



Décidons tout d'abord du point de départ de la figure. Prenons par exemple le centre. Dessinons ensuite une ligne passant par tous les points de la figure en utilisant seulement des angles droits:



Redessinons la figure sous forme de vecteurs, chacun représentant un déplacement d'une case vers le haut, le bas, la droite ou la gauche. Distinguons les vecteurs qui afficheront ou non un point sur l'écran avant de se déplacer; un cercle noir indiquant l'affichage du point:



Maintenant alignons ces vecteurs dans l'ordre où on les rencontre.



- Plaçons maintenant ces vecteurs dans la série d'octets représentant la figure en partant du vecteur A et de l'octet \emptyset . Le vecteur suivant est placé dans la case suivante disponible. Lorsque dans un octet on arrive au vecteur C et que l'on a à placer un déplacement avec affichage (1XX) ou un déplacement vers le haut, on saute ce vecteur C et on remplit le vecteur A de l'octet suivant. Enfin on remplit de " \emptyset " un dernier octet pour indiquer la fin de la figure.

Vecteur		C	B	A
Octet N°	\emptyset		↓	↓
	1		↔	↔
	2		↑	↑
	3	→	↑	↑
	4		↔	↔
	5		↓	↔
	6		↓	↓
	7		←	↓
	8			→
	9			

Vecteur	Code
↑	$\emptyset\emptyset\emptyset$
→	$\emptyset\emptyset 1$ ou $\emptyset 1$
↓	$\emptyset 1\emptyset$ ou $1\emptyset$
←	$\emptyset 1 1$ ou $1 1$
↑	$1\emptyset\emptyset$
↔	$1\emptyset 1$
↓	$1 1\emptyset$
↔	$1 1 1$

} Déplacement seul

La série d'octets de notre exemple devient alors, après remplissage des vecteurs non utilisés par des " \emptyset ".

Vecteur		C	B	A
N° octet	\emptyset	$\emptyset\emptyset$	$\emptyset 1\emptyset$	$\emptyset 1\emptyset$
	1	$\emptyset\emptyset$	$1 1 1$	$1 1 1$
	2	$\emptyset\emptyset$	$1\emptyset\emptyset$	$\emptyset\emptyset\emptyset$
	3	$\emptyset 1$	$1\emptyset\emptyset$	$1\emptyset\emptyset$
	4	$\emptyset\emptyset$	$1\emptyset 1$	$1\emptyset 1$
	5	$\emptyset\emptyset$	$\emptyset 1\emptyset$	$1\emptyset 1$
	6	$\emptyset\emptyset$	$1 1\emptyset$	$1 1\emptyset$
	7	$\emptyset\emptyset$	$\emptyset 1 1$	$1 1\emptyset$
	8	$\emptyset\emptyset$	$\emptyset\emptyset\emptyset$	$1 1 1$
	9	$\emptyset\emptyset$	$\emptyset\emptyset\emptyset$	$\emptyset\emptyset\emptyset$

Ces octets représentent chacun 8 bits "0" ou "1". Il convient alors de les représenter sous forme hexadécimale en divisant chaque octet en 2 fois 4 bits:

N° octet	0	0001	0010
1	0011	1111	
2	0010	0000	
3	0110	0100	
4	0010	1101	
5	0001	0101	
6	0011	0110	
7	0001	1110	
8	0000	0111	
9	0000	0000	

et en s'aidant de la table de codage binaire/hexadécimale suivante:

N° octet	Représentation hexadécimale
0	12
1	3F
2	20
3	64
4	2D
5	15
6	36
7	1E
8	07
9	00

binaire	hexadécimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

8 - CRÉATION D'UN FICHIER FIGURES :

Le fichier figure se compose de 3 parties:

- 1° - Le nombre de figures contenues dans le fichier
- 2° - L'adresse relative (par rapport au début du fichier) du premier octet de chaque figure, ou index
- 3° - Les figures proprement dites (la suite d'octets vue précédemment)

• Fichier figures

INDEX	Ø	Nombre de figures	§Ø à §FF (255 figures max)
	1	inutilisé	
	2	D1 MOD 256	D1 = adresse relative de la figure N° 1
	3	D1/256	
	4	D2 MOD 256	D2 = adresse relative de la figure N° 2
	5	D2/256	
	6	D3 MOD 256	D3 = adresse relative de la figure N° 3
	7	D3/256	
			D1 exprimé en hexadécimal
	D1	premier octet	
	D1+1	2ème octet	1ère figure
		§ ØØ	← fin de la 1ère figure
	D2	premier octet	
	D2+1	2ème octet	
		§ ØØ	2ème figure
	D3		← fin de la 2ème figure

Supposons que l'on veuille créer un fichier figure ne contenant que la figure élaborée dans l'exemple ci-dessus.

Le nombre de figure est alors égal à 1.

Le premier octet de la figure sera positionné immédiatement après l'index de cette figure, puisqu'il n'y a alors qu'un seul index.

L'index sera alors 04 et le fichier se présentera ainsi:

Octet N°	0	1	2	3	4	5	6	7	8	9	A	B	C	D
Contenu	01	00	04	00	12	3F	20	64	2D	15	36	1E	07	00

Nbre
de
fig.
Adresse
relative
du 1er
octet
de la
figure 1

FIGURE 1

- Le fichier figure est alors prêt à être entré en mémoire vive, à l'aide du clavier. Choisissons s'abord l'adresse hexadécimale de départ du fichier. Cette adresse doit être inférieure à la plus haute adresse mémoire disponible (\$ 4000 pour 16 K RAM, \$ 8000 32 K RAM, \$ C000 pour 48 K RAM). D'autre part la page 1 de mémoire HIRES est comprise entre \$ 2000 et \$ 3FFF. Plaçons nous légèrement en dessous, par exemple en 1DFC. Appuyer sur la touche **RESET** du clavier. Un astérisque apparaît sur l'écran, indiquant que le micro-ordinateur travaille en langage machine.

Taper:

*1DFC : 01 00 04 00 12 3F 20 64 2D 15 36 1E 07 00

Appuyer sur la touche **RETURN**. Noter les espaces entre chaque octet.

Taper: 1DFC

On voit apparaître

*1DFC - 01

ce qui indique que l'octet 01 est stocké à l'adresse mémoire 1DFC

Taper **RETURN**, il apparaît 00 04 00: l'octet 00 est stocké en 1DFD, l'octet 04 en 1DFE, etc.

Taper **RETURN**, il apparaît:

*1E00 - 12 3F 20 64 2D 15 36 1E

Ces octets sont stockés respectivement de 1E00 à 1E07.

On peut ainsi vérifier que le fichier figure est bien entré en mémoire vive.

9 - STOCKAGE D'UN FICHIER SUR BANDE MAGNÉTIQUE :

- Nous devons savoir 3 choses:

- 1° - L'adresse de départ du fichier (dans notre exemple 1DFC)
- 2° - L'adresse du dernier octet du fichier (1E09)
- 3° - La différence entre 2° et 1° (000D)

GLOSSAIRE ALPHABETIQUE DES INSTRUCTIONS ET COMMANDES BASIC ET LEUR SIGNIFICATION

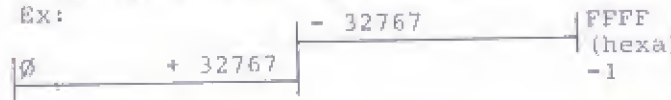
REMARQUE:

Les chiffres ou variables figurant dans les exemples ci-dessous n'ont aucune signification particulière et sont utilisés seulement à titre d'exemple.

Les Instructions se différencient des Commandes en ce qu'elles doivent être précédées d'un numéro de ligne. Elles font donc partie d'un programme comportant plusieurs instructions. Par ailleurs, plusieurs Commandes (qui sont utilisées en mode d'affichage direct) ne peuvent figurer sur une même ligne comme les Instructions multiples.

	Réf. manuel	
ABS (expr.)	page 18	donne la valeur ABSolue de l'expression entre parenthèses Ex: PRINT ABS (X)
AND	chapitre 7	opérateur logique: les deux expressions doivent être vraies toutes les deux pour que l'instruction soit vraie. Ex: 100 IF A > B AND C < D THEN 200
ASC (str g)		donne la valeur ASCII décimale de la variable alphanumérique entre parenthèses. Si la variable est composée de plus d'un caractère, seul le premier est pris en considération. Ex: PRINT ASC (Bg)
AUTO 10	page 29	indique à l'ordinateur d'afficher automatiquement les numéros de ligne du programme de 10 en 10 en commençant par la 1ère ligne du programme (pour annuler cette commande voir à MAN)
AUTO 50	page 29	identique à l'instruction ci-dessus, mais en commençant à une ligne donnée du programme (ici: 50).
AUTO 50, 10 (ou AUTO 50, 5)	page 29	indique à partir d'un n° déterminé (ici 50) que les lignes suivantes seront numérotées de 10 en 10 (ou 5 en 5).
CALL -936	chapitre 16	indique à l'ordinateur de rechercher dans sa mémoire ROM, à l'endroit

décimal spécifié par l'expression, le sous-programme correspondant.
 N.B.: les emplacements au-delà de 32767 sont affectés d'un signe négatif.
 Ex:



CLR	page 30	remet à Ø, toutes les variables existantes dans le programme en cours (sans le modifier pour autant).																
COLOR = 15	chapitre 4	<p>en mode graphique standard, cette instruction permet d'obtenir la couleur sélectionnée (15 dans l'exemple). A partir de cette instruction, tous les graphiques qui seront affichés sur l'écran, seront de la couleur choisie. Ci-dessous, code des couleurs:</p> <table><tr><td>Ø: noir</td><td>8: marron</td></tr><tr><td>1: bleu outremer</td><td>9: bleu clair</td></tr><tr><td>2: vert bouteille</td><td>10: vert pomme</td></tr><tr><td>3: bleu océan</td><td>11: turquoise</td></tr><tr><td>4: rouge foncé</td><td>12: rouge clair</td></tr><tr><td>5: violet</td><td>13: vieux rose</td></tr><tr><td>6: ocre</td><td>14: jaune</td></tr><tr><td>7: mauve</td><td>15: blanc</td></tr></table>	Ø: noir	8: marron	1: bleu outremer	9: bleu clair	2: vert bouteille	10: vert pomme	3: bleu océan	11: turquoise	4: rouge foncé	12: rouge clair	5: violet	13: vieux rose	6: ocre	14: jaune	7: mauve	15: blanc
Ø: noir	8: marron																	
1: bleu outremer	9: bleu clair																	
2: vert bouteille	10: vert pomme																	
3: bleu océan	11: turquoise																	
4: rouge foncé	12: rouge clair																	
5: violet	13: vieux rose																	
6: ocre	14: jaune																	
7: mauve	15: blanc																	
CON	pages 28 & 111	<p>utilisé pour CONTINUER l'exécution du programme en cours, après un arrêt causé par un CTRL. Ne modifie pas la valeur des variables utilisées.</p> <div style="border: 1px solid black; width: 30px; height: 15px; margin: 0 auto; text-align: center; line-height: 15px;">C</div>																
DEL 10	page 28	supprime la ligne 10 (ou n'importe quel n° ...).																
DEL 10, 50	page 28	supprime la partie du programme comprise depuis la ligne 10 (incluse) jusqu'à la ligne 50 (incluse).																
DIM	chapitre 12 chapitre 13	<p>cette instruction indique à l'appareil de réserver de la mémoire pour les variables spécifiées. Pour les registres numériques, il réserve environ 2 fois l'expression en octets (ou bytes) de mémoire (limité, bien entendu, par la mémoire disponible). Pour les registres alphanumériques, la longueur de la variable (en caractères) doit être dans la gamme de Ø à 255.</p> <p>D'autre part, seule la dernière variable peut être redéfinie à n'importe quel</p>																

		moment après son inscription. Ex: DIM A (20) DIM B\$ (100)
DSP X	page 88	fonction de dépistage utilisée pour l'affichage des valeurs que prend la variable spécifiée au cours de l'exécution du programme. Chaque nouvelle valeur apparaît avec le numéro de ligne où elle intervient. <u>Utilité principale</u> : mise au point et contrôle du programme.
END	chapitre 5	arrête le déroulement du programme. Cette instruction a le même effet que la touche RETURN .
FOR... TO... STEP... (1) (2) (3) NEXT	chapitre 9	Utilisé séquentiellement pour l'exécution d'une boucle. FOR initialise une variable à la valeur de l'expression 1, puis l'augmente de la valeur de l'expression 3 (STEP) chaque fois que l'instruction NEXT est rencontrée et ainsi de suite jusqu'à ce que la valeur de l'expression 2 (TO) soit atteinte. Si STEP n'est pas mentionnée, l'augmentation s'effectuera par pas de 1 à chaque passage. Les nombres négatifs sont autorisés. Ex: 10 FOR A = 1 TO 15 STEP 2 : NEXT A
GOSUB expr.	chapitre 14	quand elle est rencontrée, cette instruction envoie directement au sous-programme correspondant au n° de ligne précisé dans l'expression. Ex: 100 GOSUB 3000 (Attention: le maximum de sous-programmes différents pouvant se suivre avant de revenir au programme principal est de 16).
GOTO expr.	chapitre 6	même effet que ci-dessus mais à l'intérieur du programme principal. Ex: 10 GOTO 50
GR	chapitre 4	affiche le mode graphique mixte, couleurs. Délimite un écran de 40 X 40, permettant ainsi l'utilisation des 15 couleurs disponibles (le 0 = noir étant initialisé automatiquement sur la surface disponible). Le mode mixte autorise l'affichage de texte sur 4 lignes (numéros 21 à 24), de 40 caractères

		chacune, dans le bas de l'écran.
HIMEM: expr.		<p>établit la limite supérieure (emplacement <u>décimal</u> correspondant au nombre indiqué après HIMEM) de la capacité mémoire que l'on souhaite obtenir. HIMEM est réglé automatiquement sur <u>32767</u> (capacité maximum) par la commande <u>CTRL B</u>.</p> <p>Attention: ne pas omettre les deux points après HIMEM avant d'inscrire le nombre désiré.</p>
HLIN 0,39 AT 20	chapitre 4	<p>en mode graphique couleurs, cette instruction permet de tracer une ligne horizontale (dans une couleur définie au préalable) partant de la position 0 (dans notre exemple), se terminant à la position 39 et située à la position verticale déterminée ici par AT 20. Les deux positions horizontales doivent se situer dans la gamme de 0 à 39, la première étant évidemment inférieure à la seconde.</p> <p>Ex: HLIN 0,19 AT 0 affichera une ligne horizontale en haut de l'écran, partant de l'extrême gauche (et du haut de l'écran) et s'arrêtant au centre de celui-ci.</p> <p>HLIN 20, 39 AT 39 donnera une ligne horizontale en bas de l'écran, partant de son centre et s'arrêtant à l'extrême droite.</p>
IF... THEN	chapitre 6	<p><u>SI</u> (IF) l'expression est vraie, <u>ALORS</u> (THEN) l'instruction sera exécutée; si elle est fausse non exécution et passage à la ligne suivante.</p> <p>N.B.: si l'expression qui suit THEN est le n° d'une ligne du programme (ex: IF X > 1 THEN 100) la commande GOTO est sous-entendue et il n'est pas nécessaire de l'inscrire.</p>
INPUT	chapitre 11	<p>permet d'entrer des données variables par le clavier. S'il s'agit d'un nombre, un point d'interrogation sera affiché sur l'écran; il n'y en aura pas s'il s'agit d'une variable alphanumérique. On peut entrer plusieurs nombres (quantité déterminée au préalable) sur le même INPUT. Il faut alors les séparer soit par une virgule, soit en appuyant chaque fois sur <u>RETURN</u>.</p>

Par contre, les entrées de variables alphanumériques doivent obligatoirement être suivies d'un `RETURN`. Si vous désirez afficher un texte après `INPUT`, il est nécessaire de le placer entre guillemets " ".

Ex: `INPUT X, Y, Z`

`INPUT "MONTANT", FRS`

`INPUT "OUI OU NON", KAPPA`

`IN # expr.`

indique à l'ordinateur de se relier à un connecteur de périphérique E/S, au numéro spécifié par l'expression, pour obtenir des sources d'information extérieures (disquette par exemple). Chiffres possibles: 1 à 7 (le connecteur 0 n'est pas adressable depuis le BASIC. Il est utilisé pour déconnecter un périphérique et renvoyer le contrôle au clavier. Ex: `IN # 0`).

`LEN (B$)`

chapitre 12

donne la "longueur" existante (c'est-à-dire le nombre de caractères, blancs compris) de la variable alphanumérique exprimée entre parenthèses.

`LET`

opérateur d'assignation. `LET` est optionnel.

Ex: `10 LET A = 5`

`LIST`

pages 28 &
26

cette commande a pour effet de faire défiler tout le programme en mémoire sur l'écran.

`LIST 100`

page 28

affiche seulement la ligne désignée sur l'écran.

`LIST 100, 200`

page 28

affiche la partie du programme comprise entre les lignes 100 et 200 (incluses).

`LOAD`

pages 19 &
20

charge un programme BASIC existant sur une cassette magnétophone dans la mémoire RAM de l'ordinateur.

Tapez `LOAD`, puis enfoncez la touche `START` du magnétophone. Dès l'audition du sifflement du début de bande, enfoncez la touche `RETURN` de l'ordinateur. 1 "BIP" de début, 1 "BIP" de fin et l'affichage de la "pointe de flèche" indiquent une transmission correcte. Les messages `ERR` ou `MEM FULL ERR` signalent un mauvais enregistrement provenant

		soit de la bande, soit d'une faible performance (ou d'un mauvais réglage) du magnétophone.
LOMEM : expr.	pages 116 & 119-120	(voir HIMEM) établit la limite inférieure de la capacité mémoire dont vous voulez disposer pour un programme BASIC. LOMEM est réglé automatiquement à la limite minimale, soit 2048, par la commande CTRL-B . <ul style="list-style-type: none"> Attention: comme pour HIMEM, ne pas omettre les deux points après LOMEM.
MAN	page 29	annule la fonction AUTO de numérotation des lignes. Pour passer en commande MANuelle, frappez un CTRL-X , tapez les lettres MAN, puis appuyez sur RETURN .
MOD	pages 13, 43 & 81	symbole mathématique pour calculs (MODULO): exprime le reste après la division d'une première expression par une deuxième. Ex: PRINT 100 MOD 17 (résultat = 15)
NEW	page 28	annule le programme en cours. N.B.: si dans le programme que vous désirez annuler, vous avez utilisé LOMEM et HIMEM à des valeurs données, il est préférable de replacer la mémoire à sa capacité de base en faisant un RESET , puis un CTRL-B .
NEXT	chapitre 9	(voir FOR... TO) s'utilise impérativement après FOR... TO pour boucler la boucle. A chaque passage, NEXT augmente de 1 la valeur de la variable déterminée, jusqu'à la limite précisée.
NO DSP A	page 88	annule la fonction DSP pour la variable considérée (ici: A).
NOT	chapitre 7	négation logique de l'expression qui la suit: 0 si l'expression est vraie (un); 1 si l'expression est fausse (zéro) Ex: 10 IF A NOT B THEN 50
NO TRACE	page 91	annule la fonction TRACE (voir à ce mot).
OR	chapitre 7	opérateur logique: si l'une ou l'autre des deux expressions est vraie, ou si les deux sont vraies, l'ensemble est vrai. Ex: 100 IF A > B OR C < D THEN 200

Réf. manuel

PDL (expr.)	chapitre 15	donne un nombre compris entre 0 et 255, représentant la position exacte sur l'écran de la "raquette" ou du "levier" utilisé pour les jeux ou le dessin. Le n° entre parenthèses qui suit PDL désigne le n° de la "raquette" utilisée. Attention, seulement autorisés: 0, 1, 2 et 3 Ex: PRINT PDL (1)
PEEK (expr.)	chapitre 16	donne la valeur décimale du nombre stocké à l'emplacement mémoire (décimal) spécifié par le chiffre entre parenthèses. Pour les emplacements mémoire situés au-delà de 32767, utiliser les nombres négatifs. Par exemple: l'emplacement en valeur décimale de l'emplacement hexadécimal FFF0 est -16.
PLOT expr.1, expr.2	chap. 4	en mode graphique couleurs, PLOT sert à inscrire un petit rectangle (de couleur prédéterminée) à l'emplacement horizontal défini par la première expression, et à l'emplacement vertical défini par la deuxième expression, toujours de 0 à 39 pour l'horizontal et de 0 à 39 (ou de 0 à 47 en mode graphique intégral) sur le plan vertical. Ex: PLOT 19,19 affichera un petit rectangle de couleur au milieu de l'écran et PLOT 0,0 au coin supérieur gauche de l'écran.
POKE expr.1, expr.2	chap. 16	mémorise le nombre <u>décimal</u> défini par l'expression 2 (gamme de 0 à 255) à l'emplacement mémoire <u>décimal</u> déterminé par l'expression 1 Comme pour PEEK, les emplacements situés au-delà de 32767 sont affectés d'un signe négatif. Ex: POKE 50, 127 (permet l'inversion de la vidéo: écriture noire sur fond blanc)
POP	page 101	cette commande placée à la fin d'un ensemble de sous-programmes (GOSUB) permet de sauter un niveau de retour (donc de revenir, par exemple, directement au programme principal sans repasser par le sous-programme précédent).
PRINT	pages 9, 10,	affiche les données relatives aux 11... variables alpha ou alphanumériques, à

		l'emplacement situé immédiatement après celui du curseur. On peut séparer les variables soit par une virgule - auquel cas l'affichage s'effectuera en 5 colonnes d'égale grandeur - ou séparées par un point-virgule - auquel cas il n'y a aucun espacement dans l'affichage.
PR # expr.		nous avons vu que l'instruction IN # indique à l'ordinateur de se brancher à un connecteur de périphérique E/S, au n° spécifié par l'expression, pour transférer des informations de l'ordinateur vers un périphérique (chiffres possibles 1 à 7). Inversement, PR # déconnecte le périphérique et renvoie l'affichage sur l'écran.
REM	page 37	aucune action sur le programme! Utilisé surtout pour expliciter un programme d'une certaine complexité en donnant les explications des instructions qui vont suivre.
<div>RESET</div>	chapitre 2	le fait d'enfoncer cette touche, cause les faits suivants: <ul style="list-style-type: none"> - interrompt le déroulement du programme - établit le mode texte, et place l'écran à sa surface maximum - l'ordinateur passe en système MONITEUR (accès direct au langage machine), un astérisque apparaît suivi du curseur et un "BIP" résonne. <p>N.B.: le fait d'appuyer sur <div>RESET</div> ne détruit en aucun cas le programme BASIC en cours! (pour retrouver le BASIC frapper <div>CTRL C</div>).</p>
RETURN	chapitre 14	attention! <u>NE PAS CONFONDRE</u> cette instruction qui doit être tapée sur le clavier <u>lettre par lettre</u> , et qui est utilisée pour terminer un sous-programme (adressé par GOSUB), avec la <u>TOUCHE</u> <div>RETURN</div> .
RND (expr.)	page 18	donne un nombre aléatoire compris entre 0 et l'expression entre parenthèses -1, si l'expression est positive. Si elle est négative, le résultat sera compris entre 0 et l'expression entre parenthèses + 1. Ex: RND (9)=0 ou 1,2,3,4,5,6,7,8. RND(-9)=0 ou -1,-2,-3,-4,-5,-6,-7,-8.

	<u>Réf. manuel</u>	
RUN	chapitre 5	remet les variables à \emptyset , ainsi que les dimensions des registres, et exécute le programme au plus bas n° de ligne rencontré.
RUN expr.	page 28	remet les variables à \emptyset , et exécute le programme à partir du n° de ligne défini par l'expression.
SAVE	page 21	permet d'enregistrer sur une cassette un programme BASIC existant. Pour ce faire taper SAVE, enfoncer ensemble les touches RECORD et START du magnétophone, puis appuyer sur la touche RETURN du clavier.
SCRN (expr.1, expr.2)	chap. 4 & page 107	donne la couleur (nombre compris entre \emptyset et 15) de l'écran à l'emplacement défini horizontalement par l'expression n° 1 et verticalement par l'expression n° 2. Gamme comprise entre \emptyset et 47 sur le plan vertical si l'on se trouve en mode graphique intégral.
SGN (expr.)	pages 18-19 & 37	donne le signe de l'expression entre parenthèses, c'est-à-dire -1 si c'est négatif, + 1 si c'est positif, \emptyset si cette expression est égale à \emptyset .
TAB expr.	page 61	déplace le curseur à la position horizontale exacte, définie par l'expression dans la gamme de 1 à 40. Le déplacement s'effectue de la gauche vers la droite.
TEXT	chapitre 4 & page 109	remplace l'écran en mode TEXTE, soit un format de 24 lignes de 40 caractères chacune.
TRACE	page 91	fonction de contrôle et de dépistage utilisée pour afficher les numéros de ligne de chaque instruction du programme dans leur ordre exact d'exécution.
VLIN 0,39 AT 20	chapitre 4	même instruction et même effet que HLIN, mais sur le plan vertical. En mode graphique intégral, peut aller de \emptyset à 47.
VTAB	page 61	similaire à TAB, mais effectue le déplacement sur le plan vertical, dans la gamme de 1 à 24.

MESSAGES BASIC INDIQUANT LES ERREURS POSSIBLES

*** SYNTAX ERR	résulte d'une erreur de syntaxe ou de frappe. L'ordinateur vous signifie "je ne comprends pas!" Retapez l'instruction correctement Ex: PRINT = SYNTAX ERR
*** > 32767 ERR	obtenu si l'on tente d'entrer une valeur (ou un résultat d'une opération inférieure à -32767 ou supérieure à +32767.
*** > 255 ERR	obtenu si l'on a entré une valeur supérieure à 255 (dans la gamme de 0 à 255) Ex: instructions POKE
*** BAD BRANCH ERR	résulte d'une tentative de relier une instruction à un numéro de ligne in-existant.
*** BAD NEXT ERR	résulte d'une tentative d'exécuter une instruction "NEXT" pour laquelle il n'y avait aucune instruction "FOR".
*** BAD RETURN ERR	résulte d'une tentative d'exécuter plus de RETURN qu'il n'y a de sous-programmes existants (adressés par un GOSUB).
*** 16 GOSUBS ERR	le nombre de GOSUB (sous-programmes) indiqués dépasse 16! (Cette erreur est très rare!).
*** 16 FORS ERR	le nombre de boucles (FOR... TO... NEXT) indiquées dépasse 16!
*** NO END ERR	la dernière instruction exécutée dans le programme n'était pas "END".
*** MEM FULL ERR	tentative d'entrer un programme dont l'importance excède la quantité de mémoire RAM disponible. Ce message est également affiché si le volume du magnétophone est mal réglé au cours du transfert d'un programme d'une cassette dans l'ordinateur.
*** TOO LONG ERR	obtenu si l'on tente d'entrer sur le même n° de ligne plus de 12 expressions entre parenthèses ou plus de 128 caractères.

*** DIM ERR

résulte d'une tentative d'établir les dimensions d'un registre ou d'une chaîne de caractères déjà dimensionnés auparavant.

*** RANGE ERR

ERREUR de GAMME:

A - un registre a dépassé la valeur fixée par DIM

B - un registre a une dimension de 1

C - la valeur affectée à HLIN, VLIN, PLOT, TAB ou VTAB ne se trouve pas dans les limites de leurs gammes.

*** STR OVFL ERR

le nombre de caractères attribués à une variable alphanumérique a dépassé la valeur prévue par DIM.

*** STRING ERR

pratiquement toutes les erreurs concernant une variable alphanumérique (STRING) provoquent l'affichage de ce message.

RETYPE LINE

notez que ce message n'est jamais précédé des 3 astérisques. Il est obtenu quand, après un INPUT, les entrées ne correspondent pas à ce qui est attendu. Par exemple: taper des chiffres alors que des lettres sont demandées ou vice versa. L'affichage de ce message n'interrompt nullement le programme mais il implique de taper de nouvelles données, correctes cette fois.

L'affichage de la barre oblique, suivi de plusieurs "BIP" d'avertissement, vous indique que vous avez tapé une ligne contenant plus de 255 caractères. l'ordinateur vous signale qu'il n'a rien pris en mémoire.

IL FAUT RETAPER LA LIGNE (en dessous de 255 caractères bien entendu!).

STOPPED AT 155

ce message apparaît lorsqu'au cours du déroulement d'un programme, un problème quelconque se pose à l'ordinateur, l'empêchant de continuer. L'indication du n° de ligne est précieuse! En effet, le simple fait de taper LIST 155 et d'examiner l'instruction vous permet dans la plupart des cas de résoudre le problème.

COMMANDES DE LA TOUCHE "CTRL" (CONTRÔLE)

CTRL

B

page 8

utilisé quand vous êtes en système MONITEUR (indiqué par l'astérisque *) pour repasser en BASIC (indiqué par la "pointe de flèche" >).
Attention! Cela signifie que le (ou les) programme(s) BASIC existant sont DETRUITS! A également la propriété de remplacer les valeurs de LOMEM et HIMEM à leur emplacement originel (LOMEM : 2048, HIMEM : 4096 pour 4 K, 16384 pour 16 K etc...).

CTRL

C

page 28

en BASIC, arrêtez le programme et affichez le n° de ligne au moment de cet arrêt. Cependant, si à cet endroit une entrée des données est demandée (INPUT), il faut également frapper sur la touche RETURN, après le CTRL-C.
On peut reprendre l'exécution du programme par la commande CON(tinue).
Si l'ordinateur est en système moniteur (astérisque *) le fait de frapper CTRL-C et RETURN réinitialise le BASIC SANS DETRUIRE le programme en cours

CTRL

G

pages 7 &
110

émet un son (BIP)

CTRL

H

fait revenir le curseur en arrière en effaçant les caractères (qui sont visibles sur l'écran) de la mémoire de l'ordinateur mais non de l'écran.
La touche ← figurant sur votre clavier, accomplit la même fonction.

CTRL

J

permet de vider l'affichage écran par le haut tout en conservant la position curseur à l'endroit du départ.

CTRL

V

inverse de CTRL-H. Déplace le curseur de la gauche vers la droite et n'a aucun effet sur les caractères affichés soit en mémoire soit sur l'écran.
La touche → du clavier accomplit la même fonction.

CTRL

pages 28 &
29

efface complètement la ligne en cours:

X

- 1° - de l'écran
- 2° - de la mémoire

COMMANDES DE LA TOUCHE "ESC" (ÉCHAPPEMENT)

ESC

page 30

déplace le curseur vers la droite

A

ESC

page 30

déplace le curseur vers la gauche

B

ESC

page 30

déplace le curseur vers le bas

C

ESC

pages 28 &
30

déplace le curseur vers le haut

D

ESC

efface le texte entre le curseur et la
fin de la ligne

E

ESC

efface le texte entre le curseur et la
fin de la page

F

ESC

page 7

déplace le curseur au coin supérieur
gauche de l'écran et efface toutes les
inscriptions de l'écran.

OPÉRATEURS BASIC

Symbole:		Désignation:
()	page 15	Dans une série d'opérations, les expressions entre parenthèses sont toujours calculées en priorité. Ex: $A = 4 * (2 + 2)$ soit résultat 16.
\wedge	page 13	indique l'élévation à une puissance. Ex: $4 \wedge 4$ (soit 256) N.B.: symbole obtenu en appuyant simultanément sur SHIFT et N .
*	chapitre 3	Symbole de la multiplication. Attention: ne pas utiliser les notations mathématiques habituelles, telles que $(2+3)(4)$. Pour obtenir le bon résultat par l'ordinateur, il faudrait écrire dans ce cas $(2+3) * 4$.
/	chapitre 3	Symbole de la division.
+	chapitre 3	Symbole de l'addition. Peut être placé devant une expression et dans ce cas lui assigne une valeur positive.
-	chapitre 3	Symbole de la soustraction. Peut être également placé devant une expression et dans ce cas lui assigne une valeur négative.
=	chapitre 3	affecte une valeur à une variable Ex: $A = 5$ OU $KAPPA\delta = \text{"BONJOUR"}$
" "	page 9	Guillemets. Utilisés pour afficher une proposition quelconque sur l'écran Ex: <code>PRINT "BONJOUR"</code>
,	chapitre 10	Virgule. Utilisée après un <code>PRINT</code> ou un <code>INPUT</code> (voir explication détaillée au chapitre X).
;	chapitre 10	Point-virgule (voir explication détaillée au chapitre X).
:	page 91	Deux points. Ils permettent de placer des instructions multiples sur une seule ligne.

ANNEXE: PROGRAMMES DIVERS

MASTER MIND

Un joueur joue contre l'ordinateur. Les règles du jeu sont les suivantes:
la machine choisit 5 couleurs, d'une manière aléatoire, et les garde en mémoire.

Le joueur choisit 5 couleurs sur un choix de 9 possibles, et l'ordinateur les compare à celles qu'il a en mémoire. Puis à chaque nouveau choix, il affiche le nombre de couleurs trouvées dans l'ordre, puis dans le désordre jusqu'à ce que le résultat soit exact.

La patience de l'ordinateur ne va que jusqu'à 14 coups...

```
10  TEXT : CALL -936 : DIM A (10), B (10), C (10), D (10), AS (10)
20  VTAB 10 : TAB 11 : PRINT "JEU DE MASTER MIND"
30  PRINT : TAB 11 : PRINT " _____ "
40  FOR A = 0 TO 800 : NEXT A
50  CALL -936 : GR : Y = 1 : VTAB 21
60  PRINT "  1 2 3 4 5 6 7 8 9"
70  FOR X = 1 TO 9 : COLOR = X : PLOT 2 * X, 39 : NEXT X
80  POKE 34, 21 : POKE 35, 24
90  FOR X = 1 TO 5 : A (X) = RND (9) + 1 : NEXT X
100 M = 0 : N = 0 : POKE 35, 24
110 INPUT C(1), C(2), C(3), C(4), C(5) : FOR X = 1 TO 5 : COLOR = C(X) :
    PLOT 2 * X - 1, Y : NEXT X
120 FOR X = 1 TO 5 : B(X) = 0 : D(X) = 0 : NEXT X : FOR X = 1 TO 5 : IF
    A(X) < > C(X) THEN 140
130 B(X) = 1 : D(X) = 1 : M = M + 1 : COLOR = 4 : PLOT 14 + 2 * M, Y
140 NEXT X
150 IF M = 5 THEN 280
160 FOR X = 1 TO 5 : IF B(X) = 1 THEN 220
170 FOR V = 1 TO 5 : IF D(V) = 1 THEN 210
180 IF B(X) = 1 THEN 220
190 IF A(X) < > C(V) THEN 210
200 B(X) = 1 : D(V) = 1 : N = N + 1 : COLOR = 6 : PLOT 27 + 2 * N, Y
210 NEXT V
220 NEXT X
230 Y = Y + 2
240 IF Y < 30 THEN 100
250 FOR X = 1 TO 5 : COLOR = A(X) : PLOT 2 * X - 1, Y : NEXT X
```

```
260 INPUT "VOULEZ-VOUS REJOUER? OUI/NON", A$  
270 IF A$ = "NON" THEN 290 : GOTO 50  
280 TAB 25 : PRINT "B R A V O !!!" : PRINT : GOTO 260  
290 END
```

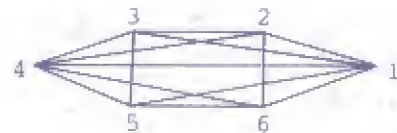

TRIANGLE

Vous jouez contre l'ordinateur!

Données: 6 points, numérotés de 1 à 6



Vous et l'ordinateur tirez un trait d'un point à un autre à tour de rôle.
Il y a 15 possibilités



1,2 - 1,3 - 1,4 - 1,5 - 1,6 -
2,3 - 2,4 - 2,5 - 2,6 -
3,4 - 3,5 - 3,6 -
4,5 - 4,6 -
5,6 -

Les traits des deux joueurs sont de couleurs différentes.
Il est interdit de repasser sur un trait qui existe déjà!

But: éviter de réaliser un triangle. Le joueur qui forme le premier triangle avec ses traits a perdu.

```

10  DIM X (6), Y (6), M (8), L (8), N (8), E (15), F (15), C (15): X (1)
    = 35 : X (2) = 27: X (3) = 11 : X (4) = 3
20  X (5) = 11 : X (6) = 27 : Y (1) = 19 : Y (2) = 7 : Y (3) = 7 : Y (4)
    = 19 : Y (5) = 31: Y (6) = 31
30  C (1) = 12 : C (2) = 23 : C (3) = 34 : C (4) = 45 : C (5) = 56 :
    C (6) = 16 : C (7) = 13
40  C (8) = 24 : C (9) = 35 : C (10) = 46 : C (11) = 15 : C (12) = 26:
    C (13) = 14 : C (14) = 25 : C (15) = 36
50  E (1) = -2 : E (2) = -1 : E (3) = -2 : E (4) = 2 : E (5) = 1 : E (6)
    = -2 : E (7) = -2
55  E (8) = -2 : E (9) = 0 : E (10) = 2 : E (11) = -2 : E (12) = 0 :
    E (13) = -1 : E (14) = -2 : E (15) = 2
60  F (1) = -3 : F (2) = 0 : F (3) = 3 : F (4) = 3 : F (5) = 0 : F (6)
    = 3 : F (7) = -1
65  F (8) = 1 : F (9) = 1 : F (10) = 1 : F (11) = 1 : F (12) = 1 :
    F (13) = 0 : F (14) = 3 : F (15) = 3
70  GR : COLOR = 15 : FOR I = 1 TO 6 : PLOT X (I), Y (I) : NEXT I
80  G = 0 : P = 14 : FOR J = 1 TO 8 : M (J) = 0 : L (J) = 0 : N (J)
    = 0 : NEXT J
100 PRINT "A VOUS" : INPUT A, B : G = G + 1
    
```

```

120 FOR K = 1 TO G : IF A * 10 + B = M (K) THEN 100 : NEXT K : FOR L
    = 1 TO 8 : N (L) = M (L) : NEXT L
140 FOR O = 1 TO 7 : IF N (O) = 0 THEN 220
150 IF N (O) * 10 ≠ A THEN 210
160 U = B : V = N (O) MOD 10 : FOR R = 1 TO ABS (G) : IF U > V THEN 190:
    GOTO 200
190 Q = U : U = V : V = Q
200 IF U * 10 + V = N (R) THEN 300 : NEXT R
210 NEXT O
220 FOR S = 1 TO 7 : IF N (S) MOD 10 ≠ A THEN 280
230 T = B : W = W(S)/10 : FOR H = 1 TO ABS (G) : IF T > W THEN 260 :
    GOTO 270
260 E = T : T = W : W = E
270 IF T * 10 + W = N (H) THEN 300 : NEXT H
275 IF G = -1 THEN 330
280 IF S = G OR S = -G - 1 THEN 330
290 NEXT S : GOTO 330
300 IF G < 0 THEN 410
310 PRINT "PERDU" : GOTO 460
320 PRINT "GAGNE" : GOTO 460
330 D = A * 10 + B : C = 2 : IF G > 0 THEN C = 1 : COLOR = C : Y = Y(A)
335 FOR A1 = 1 TO 15 : IF C (A1) ≠ D THEN NEXT A1 : D = A1
340 FOR Z = X (A) TO X (B) STEP E (D) : PLOT Z, Y : IF Z = X (B) AND
    (X (A) ≠ X (B)) THEN 360
350 PLOT Z + E (D)/2, Y + F (D)/3 : Y = Y + F (D) : IF Y > 31 THEN 360:
    NEXT Z
360 G = -G : P = P - 1
370 IF G = 0 THEN 390 : M (-G) = A * 10 + B : GOTO 400
390 L (G) = A * 10 + B : GOTO 100
400 N = 0 : GOTO 420
410 N = N + 1 : IF N > 200 THEN 320 : IF N > 25 THEN PRINT "JE CHERCHE"
420 A1 = RND (15) + 1 : CH = C (A1)
430 FOR X = 1 TO -G : IF CH = M (X) OR CH = L (X) OR CH = A * 10 + B
    THEN 410 : NEXT X
440 A = CH/10 : B = CH MOD 10 : FOR M = 1 TO 8 : N (M) = L (M) : NEXT M:
    GOTO 140
460 IF PEEK (-16384) > 127 THEN 70 : POKE -16368, 0 : GOTO 460

```

Pour recommencer à jouer, tapez sur une touche quelconque du clavier.

PROGRAMME GENERANT DES MOTS ALEATOIRES

Notes

146

```

10  DIM ALPHABET$ (100), SIGNE$ (100), MOT$ (10), SIGMOT$ (10), B$ (10),
    C$ (50)
20  COMPTEUR = 1 : M = 1
30  ALPHABET$ = "AAAAAAAAABBCDDDEEEEEEEEEEEEEEEFFGGHHIIJJKLLLLLLMMNN
    NNNDDDDDDOPPPRRRRRRSSSSSSSTTTTTUUUUUUUVVWXYZ"
40  SIGNE$ = "111111112222222111111111111111222233113333222222222
    2222111111223222222222222222222111111333333"
50  C$ = "BLBRCLCRDRFLFRGLGRPLPRSCSPSTTRCTRDGRNBNDNSNGNTGN"
60  LONGUEUR = RND (5) + 2
70  IF COMPTEUR + LONGUEUR + 1 < 40 THEN 90
80  COMPTEUR = 1 : PRINT : PRINT
90  FOR N = 1 TO LONGUEUR
100 L = RND (LEN (ALPHABET$)) + 1
110 MOT$ (N) = ALPHABET$ (L, L)
120 SIGMOT$ (N) = SIGNE$ (L, L)
130 IF N = 1 THEN 220
140 IF SIGMOT$ (N, N) = SIGMOT$ (N - 1, N - 1) THEN 180
150 B$ = "2332"
160 IF SIGMOT$ (N - 1, N) = B$ (1, 2) OR SIGMOT$ (N - 1, N) = B$ (3, 4)
    THEN 500 : GOTO 220
170 M = 1 : GOTO 220
180 M = M + 1 : B$ = "3"
190 IF SIGMOT$ (N, N) ≠ B$ (1, 1) THEN 210
200 M = M - 1 : GOTO 100
210 IF M = 2 THEN 300 : GOTO 200
220 NEXT N
230 PRINT MOT$:
240 PRINT " ";
250 COMPTEUR = COMPTEUR + LONGUEUR + 1 : GOTO 60
300 B$ = "1" : IF SIGMOT$ (N, N) ≠ B$ THEN 390
310 B$ = "EE"
320 IF MOT$ (N - 1, N) ≠ B$ THEN 340
330 IF N ≠ LONGUEUR THEN 200 : GOTO 220
340 B$ = "OUAUEUOI"
350 FOR X = 1 TO 7 STEP 2
360 IF MOT$ (N - 1, N) = B$ (X, X + 1) THEN 220
    
```

```
370 NEXT X : GOTO 200
390 IF N ≠ 2 THEN 430
400 FOR X = 1 TO 29 STEP 2
410 IF MOTS (N - 1, N) = CS (X, X + 1) THEN 220
420 NEXT X : GOTO 200
430 IF MOTS (N - 1, N - 1) = MOTS (N, N) THEN 220
450 FOR X = 1 TO 47 STEP 2
460 IF MOTS (N - 1, N) = CS (X, X + 1) THEN 220
470 NEXT N : GOTO 200
500 BS = "CHNJNQ"
510 FOR X = 1 TO 5 STEP 2
520 IF MOTS (N - 1, N) = BS (X, X + 1) THEN 220
530 NEXT X : GOTO 200
```


Adresse en hexa	Mémoire en kilo octet	Adressage en décimal	Utilisation	
FFFF	64 K	- 1	Programmes: - MONITEUR (en ROM) - BASIC (en ROM)	
F000		- 8192		
	56 K		Mémoire libre, prévue pour extensions futures (PROMS)	
D000		- 12288		
	52 K		Programme d'entrée/sortie pour périphériques	
C000		- 16384		
	48 K		sans Graphique à haute résolution	avec Graphique à haute résolution
8001 8000		- 32767 -	Espace disponible utilisateur	Espace disponible utilisateur
7FFF	32 K	32767		
4000		16384	(Limite donnée par capacité de mémoire)	Mémoire écran: haute résolution
	16 K			
2000		8192	Espace disponible utilisateur	Espace disponible utilisateur
	8 K			
1000		4096	Programme initialisation haute résolution	Programme initialisation haute résolution
0C00	4 K	3072		
	3 K		Espace disponible utilisateur	Espace disponible utilisateur
0800		2048		

	2 K		Mémoire écran: texte ou graphisme couleur
Ø4ØØ		1Ø24	
Ø3FF	1 K	1Ø23	
Ø		Ø	Espace de travail



FRANCE

ITT SOCIÉTÉ
97 Avenue de Verdun
91230 - ROMORVILLE

BELGIUM

BELL TELEPHONE
2440 - CHAL
BRUXELLES